НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.414.38

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Спосіб автоматизації процесу створення специфікацій варіантів використання на основі їх атрибутів»

Виконав:
студент ІІ курсу, групи КП-81мп
Шабіневич Роман Олександрович _____

Керівник:
Ст. викладач кафедри ПЗКС, к.т.н.,
Рибачок Н.А. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай М.В. _____

Рецензент:
Доцент кафедри ІЕ ФЕЛ, к.т.н., доцент
Вунтесмері Ю.В. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (освітня програма) – 121 «Інженерія програмного забезпечення» ("Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем")

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студентові

Шабіневичу Роману Олександровичу

1. Тема дисертації «Спосіб автоматизації процесу створення специфікацій варіантів використання на основі їх атрибутів», науковий керівник дисертації Рибачок Наталія Антонівна, к.т.н., затверджені наказом по університету від «13» листопада 2019 р. № 3895-С.

2. Термін подання студентом дисертації «16» грудня 2019 р.

3. Об'єкт дослідження: процес специфікації варіантів використання.

4. Предмет дослідження: спосіб автоматизації процесу специфікації варіантів використання.

5. Перелік завдань, які потрібно розробити:
- оцінити сучасний стан проблеми, обґрунтувати актуальність напрямку досліджень, сформулювати мету та задачі дослідження;
- проаналізувати основні підходи до специфікації варіантів використання;
- проаналізувати основні шаблони специфікацій;
- описати структурні компоненти, що є складовими способу автоматизації процесу створення специфікацій варіантів використання на основі їх атрибутів;
- виконати тестування розробленого програмного забезпечення для автоматизації процесу створення специфікацій варіантів використання на основі їх атрибутів та проаналізувати отримані результати;
- описати бізнес-модель, що дозволить представити на ринку повноцінний програмний продукт із використанням представлених у роботі напрацювань.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
- ER діаграма бази даних;
- Приклади специфікацій.

7. Орієнтовний перелік публікацій:
   - Тези доповіді “Спосіб автоматизації процесу створення специфікацій варіантів використання на основі їх атрибутів”

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., к.т.н., доцент		

9. Дата видачі завдання «25» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.11.2018	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.12.2018	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.02.2019	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	05.04.2019	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.05.2019	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	15.06.2019	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2019	13.11.2019	
8.	Оформлення текстової і графічної частини магістерської дисертації	05.12.2019	

Студент Р.О. Шабіневич

Науковий керівник дисертації Н.А. Рибачок

РЕФЕРАТ

Актуальність теми. На даний момент для однозначного бачення розроблюваної системи виконавцями та замовниками застосовуються специфікації варіантів використання, оскільки звичайні вимоги не можуть достатньо деталізувати розроблювану систему. Проте специфікація варіантів використання є складним процесом, який за браком ресурсів не рідко пропускається, що призводить до того, що замовники отримують програмний продукт, який не задовольняє їх очікуванням, проте відповідає вимогам, оскільки їх можна трактувати по різному. Цю проблему частково вирішують програмні засоби для автоматизації процесу специфікації, проте вони не є універсальними та широковживаними, оскільки спрямовані на роботу з конкретною методологією. Також процес внесення змін в специфікації є достатньо складним, що також не сприяє їх використанню, оскільки далеко не завжди вимоги до розроблюваного програмного продукту є незмінними на протязі всього етапу розроблення. Тому доцільно розробити програмне рішення, яке дозволить автоматизувати процес специфікації варіантів використання на рівні вибору рівня деталізації з інтерфейсом, що адаптується до обраного рівня деталізації та модульним збереженням готових специфікацій.

Об’єктом дослідження є процес специфікації варіантів використання.

Предметом дослідження є спосіб автоматизації процедури специфікації варіантів використання.

Мета дослідження: дослідження існуючих підходів до специфікації варіантів використання, створення узагальненого способу вибору рівня деталізації, аналіз існуючих шаблонів та створення набору шаблонів який дозволить нарощувати рівень деталізації і дозволить реалізувати

модульність подання специфікацій та розробити спосіб автоматизації специфікації варіантів використання на основі їх атрибутів.

Методи дослідження. В даній роботі використовуються методи теоретичного дослідження: аналіз, синтез та узагальнення. Також застосовувалися емпіричні методи: експеримент, спостереження, вимірювання та опис.

Наукова новизна роботи полягає в наступному:

1. Розроблено узагальнений спосіб вибору рівня деталізації специфікації ВВ, який базується на використанні атрибутів. Набір атрибутів, ступінь їх градації та ваги визначаються користувачем. Це дозволяє використовувати атрибути з різних методик пріоритезації та регулювання їх вплив при виборі рівня деталізації.
2. Підібрано та модифіковано шаблони специфікацій ВВ, які дозволяють нарощувати рівень деталізації та представляти специфікації ВВ в модульному вигляді.
3. На основі отриманих узагальненого способу та шаблонів розроблено спосіб автоматизації процесу створення специфікацій варіантів використання на основі їх атрибутів.

Практична цінність отриманих в роботі результатів полягає в тому, що розроблену систему можна використовувати для автоматизації процесу специфікації варіантів використання на основі їх атрибутів, що в свою чергу дозволить зменшити часові витрати на створення специфікацій, за рахунок автоматичного вибору рівня деталізації, на основі атрибутів та автоматичного налаштування інтерфейсу відповідно до рівня деталізації. Крім того специфікації представляються в модульному вигляді, що дозволяє значно спростити супровід та при потребі, нарощувати деталізацію специфікацій.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на XII науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2019 (Київ, 13-14 листопада 2019 р.).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п'яти розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи.

У першому розділі проаналізовано існуючі підходи до специфікації варіантів використання, існуючі шаблони та програмне забезпечення для написання специфікацій.

У другому розділі детально викладено рішення, що пропонується. Модифікація підходу полягає у тому, що вибір рівня деталізації проводиться не експертним шляхом, а з допомогою узагальненого способу, який базується на використанні атрибутів, заданих користувачем. Також підібраний та модифікований набір шаблонів специфікацій, який дозволяє реалізувати нарощування деталізації специфікацій та представляти їх в модульному вигляді.

У третьому розділі описано архітектуру розробленого програмного забезпечення, детально проаналізована його функціональність та деталі реалізації.

У четвертому розділі описано методологію тестування для розроблюваної системи, проаналізовано результати роботи та визначено напрями для подальшого вдосконалення системи у майбутньому. За результатами тестування можна заявити, що розроблена система не має помилок, які можуть призвести до виключних ситуацій при роботі

користувача. Тому може вважатися готовою до залучення бета-тестувальників.

У п’ятому розділі було сформовано та побудовано бізнес-модель кінцевого продукту, що описує ключові моменти в організації діяльності, пов’язаної з поширенням розробленої системи для створення крос-платформних користувацьких додатків.

У висновках проаналізовано отримані результати роботи.

У додатках наведено повний лістинг програмної реалізації та плакати.

Робота виконана на 81 аркушах, містить 3 додатки та посилання на список використаних літературних джерел з 18 найменувань. У роботі наведено 34 рисунки, 14 таблиць та 35 лістингів.

Ключові слова: веб-додаток, специфікації варіантів використання, модульність специфікацій.

ABSTRACT

Topicality. Today, the specifications of the use cases are applied to the unambiguous vision of the developed system by contractors and customers, since the usual requirements cannot sufficiently detail the developed system. However, specification of use cases is a complex process that is seldom missed due to a lack of resources, which results in customers receiving a software product that does not meet their expectations, but meets the requirements because they can be interpreted differently. This problem is partially solved by software to automate the specification process, but they are not universal and widely used because they are designed to work with a specific methodology. Also, the process of making changes to the specification is complex enough that it is also not conducive to their use, since it is not always the requirements for the developed software product that remain unchanged throughout the development phase. Therefore, it is advisable to develop a software solution that will automate the process of specification of use cases at the level of detail level selection with an interface that adapts to the selected level of detail and modular storage of finished specifications.

Object of research is the process of specifying use cases.

Subject of research is a way to automate the procedure for specifying use cases.

Research objective exploring existing approaches to specification of use cases, creating a generalized way of selecting the level of detail, analyzing existing templates and creating a set of templates that will increase the level of detail and allow to realize the modularity of submission of specifications and develop a way to automate the specification of use cases based on their attributes.

Research methods. This paper uses methods of theoretical research: analysis, synthesis and generalization. Empirical methods were also used: experiment, observation, measurement and description.

Scientific novelty of the work is as follows:

1. A generalized method for selecting the level of detail of the BB specification based on the use of attributes is developed. The set of attributes, the degree of their gradation and the weight are determined by the user. This allows you to use attributes from different prioritization techniques and regulate their impact when choosing the level of detail.
2. UC specification templates have been selected and modified to increase the level of detail and to represent the UC specification in modular form.
3. Based on the generalized method and templates obtained, a method was developed to automate the process of creating specifications for use cases based on their attributes.

The practical value the result obtained is that the developed system can be used to automate the process of specification of use cases based on their attributes, which in turn will reduce the time spent on creating specifications, by automatically selecting the level of detail, based on attributes and automatically configuring the interface according to the level of detail. In addition, the specifications are presented in modular form, which greatly simplifies the maintenance and, if necessary, increase the detail of the specifications.

Approbation. The main provisions and results of the work were presented and discussed at the XII Scientific Conference of Undergraduate and Graduate Students in Applied Mathematics and Computing PMK-2019 (Kyiv, November 13-15, 2019).

Structure and content of the thesis. The master's thesis consists of an introduction, five sections, conclusions and appendices.

The introduction provide the general characteristics of the work are given, the current state of the problem is evaluated, the relevance of the research direction is substantiated, the purpose and tasks of the research are formulated, the scientific novelty of the obtained results and the practical value of the work are shown.

The first section existing approaches to use case specification, existing templates and specification software were analyzed.

The second section the proposed solution was detailed. The modification of the approach is that the choice of level of detail is made not by expert means, but by a generalized method based on the use of user-defined attributes. A set of specification templates has also been selected and modified to allow the specification to be expanded and presented in a modular manner.

The third section the architecture of the developed software was described, its functionality and implementation details were analyzed in detail.

The fourth section the testing methodology for the developed system was described, the results of the work were analyzed and directions for further improvement of the system were determined. According to the results of testing, it can be stated that the developed system has no errors that can lead to exceptional situations when the user operates. Therefore, it may be considered ready to engage beta testers.

In the fifth section, an end-product business model was created and built that describes the key elements in organizing activities related to the automation of the use case specification process.

The conclusion is analyzed the results of this master's work.

The applications provide auxiliary functions for manipulating the internal state of plugins, examples of platform-specific plugins.

The work is made on 81 sheets, contains 3 appliccations and links to the list of used literary sources of 18 titles. The paper presents 34 figures, 14 tables and 35 listings.

Keywords: web-platform, use case specifying, modularity of specifications.

РЕФЕРАТ

Актуальность темы. На данный момент для однозначного видения разрабатываемой системы исполнителями и заказчиками применяются спецификации вариантов использования, поскольку обычные требования не могут достаточно детализировать разрабатываемую систему. Однако спецификация вариантов использования является сложным процессом, который за неимением ресурсов нередко пропускается, что приводит к тому, что заказчики получают программный продукт, который не удовлетворяет их ожиданиям, однако соответствует требованиям, поскольку их можно трактовать по-разному. Эту проблему частично решают программные средства для автоматизации процесса спецификации, однако они не являются универсальными и широкоупотребляемыми, поскольку направлены на работу с конкретной методологии. Также процесс внесения изменений в спецификации достаточно сложным, что также не способствует их использованию, поскольку далеко не всегда требования к разрабатываемого программного продукта являются неизменными на протяжении всего этапа разработки. Поэтому целесообразно разработать программное решение, которое позволит автоматизировать процесс спецификации вариантов использования на уровне выбора уровня детализации с интерфейсом, адаптируется к выбранного уровня детализации и модульным сохранением готовых спецификаций.

Объектом исследования является процес спецификации вариантов использования.

Предметом исследования является способ автоматизации процедуры спецификации вариантов использования.

Цель исследования: исследования существующих подходов к спецификации вариантов использования, создания обобщенного способа

выбора уровня детализации, анализ существующих шаблонов и создание набора шаблонов который позволит наращивать уровень детализации и позволит реализовать модульность представления спецификаций и разработать способ автоматизации спецификации вариантов использования на основе их атрибутов.

Методы исследования. В данной работе используются методы теоретического исследования: анализ, синтез и обобщение. Также применялись эмпирические методы: эксперимент, наблюдение, измерение и описание.

Научная новизна работы заключается в следующем:

1. Разработан обобщенный способ выбора уровня детализации спецификации ВВ, основанный на использовании атрибутов. Набор атрибутов, степень их градации и веса определяются пользователем. Это позволяет использовать атрибуты из разных методик приоритезации и регулирования их влияние при выборе уровня детализации.
2. Подобраны и модифицировано шаблоны спецификаций ВВ, которые позволяют наращивать уровень детализации и представлять спецификации ВВ в модульном виде.
3. На основе полученных обобщенного образа и шаблонов разработан способ автоматизации процесса создания спецификаций вариантов использования на основе их атрибутов.

Практическая ценность полученных в работе результатов заключается в том, что разработанную систему можно использовать для автоматизации процесса спецификации вариантов использования на основе их атрибутов, в свою очередь позволит уменьшить временные затраты на создание спецификаций, за счет автоматического выбора уровня детализации, на основе атрибутов и автоматической настройки интерфейса в соответствии с уровнем детализации. Кроме того

спецификации представляются в модульном виде, что позволяет значительно упростить сопровождение и при необходимости наращивать детализацию спецификаций.

Апробация работы. Основные положения и результаты работы были представлены и обсуждались на ХИИ научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2019 (Киев, 13-15 ноября 2019).

Структура и объем работы. Магистерская диссертация состоит из введения, пяти глав, заключения и приложений.

Во введении предоставлено общую характеристику работы, выполнена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов и практическую ценность работы.

В первом разделе проанализированы существующие подходы к спецификации вариантов использования, существующие шаблоны и программное обеспечение для написания спецификаций.

Во втором разделе детально изложены решения, предлагается. Модификация подхода заключается в том, что выбор уровня детализации проводится не экспертным путем, а с помощью обобщенного способа, основанного на использовании атрибутов, заданных пользователем. Также подобранный и модифицирован набор шаблонов спецификаций, который позволяет реализовать наращивания детализации спецификаций и представлять их в модульном виде.

В третьем разделе описано архитектуру разработанного программного обеспечения, детально проанализирована его функциональность и детали реализации.

В четвертом разделе описано методологию тестирования для разрабатываемой системы, проанализированы результаты работы и

определены направления для дальнейшего совершенствования системы в будущем. По результатам тестирования можно заявить, что разработанная система не имеет ошибок, которые могут привести к исключениям при работе пользователя. Поэтому может считаться готовой к привлечению бета-тестировщиков.

В пятом разделе сформирован и построено бизнес-модель конечного продукта, описывает ключевые моменты в организации деятельности, связанной с автоматизацией процесса спецификации вариантов использования.

В выводах проанализированы полученные результаты работы.

В приложениях приведен полный листинг программной реализации и плакаты.

Работа выполнена на 81 листах, содержит 3 приложения и ссылки на список использованных литературных источников из 18 наименований. В работе приведены 34 рисунков, 14 таблиц и 35 листингов.

Ключевые слова: веб-приложение, спецификации вариантов использования, модульность спецификаций.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Варіант використання (ВВ) – послідовність певних подій, що показують як система повинна взаємодіяти з дійовими особами для досягнення їх цілей.

Атрибут – певна характеристика, яка має фіксовану градацію допустимих значенн та задається для всіх варіантів використання (наприклад складність, пріоритет та ін.). Використовується при виборі рівня деталізації опису ВВ.

Дійова особа – особа, яка бере участь у роботі сценаріїв варіантів використання.

БД – база даних;

API – Application Programming Interface – інтерфейс, який дозволяє розробникам використовувати готові блоки для побудови програми. Являється абстракцією, яка описує функціональність, яку надає певний програмний компонент;

Фреймворк – framework – це комплекс програмних рішень, що утворює певний каркас, який складається з допоміжних програм, бібліотек коду та API. Використовуються для спрощення розробки програмних продуктів та надання стандартизованого підходу до їх проектування та розгортання;

SQL – Structured Query Language – декларативна мова програмування, яка застосовується для створення, модифікації та управління даними в реляційних БД;

СКБД – Система Керування Базами Даних – сукупність програмних та лінгвістичних засобів загального або

спеціального призначення, що забезпечують управління створенням та використанням баз даних;

HTTP – HyperText Transfer Protocol – протокол прикладного рівня передачі даних;

ORM – Object-Relational Mapping – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних»;

URL – Uniform Resource Locator – це форма запису інтернет-адрес, виконана у відповідності до встановлених стандартів;

HTML – HyperText Markup Language – це основна мова, за допомогою якого створюються веб-сторінки;

CSS – Cascading Style Sheets – формальна мова опису зовнішнього вигляду документа, написаного з використанням мови розмітки.

ВСТУП

Процес специфікації варіантів використання є важливим етапом розроблення програмного забезпечення, оскільки він дозволяє отримати чітке уявлення про майбутній програмний продукт. Варіанти використання і без процесу їх специфікації включають в себе певну інформацію, що дає уявлення про можливу реалізацію. Проте, здебільшого, бачення розроблюваної системи у замовників і зацікавлених осіб не співпадає з баченням розробників, оскільки останні не в повній мірі знайомі з предметною областю та можливими нюансами роботи організації-замовника. Це суттєво підвищує ймовірність того, що програмний продукт не буде задовольняти очікуванням зацікавлених осіб чи не буде сумісним з внутрішніми процесами організації-замовника. Мінімізувати ці ризики покликана специфікація варіантів використання. Проте цей етап часто пропускається за браком часу або ресурсів. Також не останню роль у цьому відіграє велика кількість паперової роботи, адже кількість ВВ навіть в проектах середнього масштабу є досить великою. Крім того специфікації в класичному вигляді практично не підтримують внесення змін, оскільки це процес, трудомісткість якого може бути рівною написанню нової специфікації. Тому доцільно автоматизувати даний процес, зробити його більш гнучким та полегшити супровід готових специфікацій.

1. АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО СПЕЦИФІКАЦІЇ ВАРІАНТІВ ВИКОРИСТАННЯ

1.1. Варіанти використання: їх призначення та опис

Для того щоб описати поведінку системи аналітиками застосовуються варіанти використання.

Варіант Використання (ВВ, прецедент або сценарій використання) - це послідовність деяких подій, що показують як система повинна взаємодіяти з користувачами (що називаються, акторами або дійовими особами) та іншими системами для досягнення певної мети користувачів.

Характеристики варіантів використання:

- Ініціюються дійовою особою.
- Є моделлю взаємодій між дійова особою і системою.
- Описують послідовність дій.
- Містять в собі функціональні вимоги.
- Повинні мати деяке значення для дійової особи.
- Представляють повний і маючий сенс сценарій подій.

Призначення варіантів використання:

- сприяти угоді між розробниками, замовниками і користувачами на тему того, що саме система повинна робити (ВВ стає свого роду контрактом між розробниками і замовниками);
- є основою для специфікацій варіантів використання, які відіграють велику роль у проектуванні системи;
- з варіантів використання можна створювати діаграми послідовностей, діаграми зв'язків та діаграми класів;
- з варіантів використання можна отримувати документацію для користувачів;
- варіанти використання можуть також бути корисні в плануванні формального змісту ітерацій і дозволяють розробникам системи краще зрозуміти призначення системи;

- можна використовувати в якості вхідних даних для тестових сценаріїв (test cases).

Варіанти використання є інструментом для документації потенційних вимог при створення нової програмної системи або зміни існуючої.

Актори та їх цілі

Варіант використання призначений для досягнення цілей користувача при взаємодії із системою. Тому очевидно, що основними характеристиками ВВ є користувач та його ціль.

Основний потік, альтернативи, розширення, виключні ситуації

Особливості взаємодії актора і системи в межах певного ВВ описується так званими сценаріями варіантів використання. Сценарій, який є ідеальним і сприяє досягненню цілі актора називається основним (базовим).

Крім єдиного основного сценарію виконання ВВ, який закінчується досягненням цілі актора (рис. 1), ВВ може містити альтернативні потоки (рис. 2), виключні ситуації (рис. 3) та розширення (рис. 4).

Передумови та післяумови

Передумови – це стан, в якому повинна знаходитися система та її оточення перед початком варіанту використання. Предумови повинні бути такі, щоб система могла забезпечити їх істинність. Вони описуються в специфікації і система не перевіряє їх при виконанні варіанту використання. Передумов для кожного ВВ може бути декілька.

Післяумови – це стан, в якому повинна знаходитися система після закінчення варіанту використання. Післяумов для одного варіанту використання може бути декілька. Вони вказуються як для основного (успішного) потоку, так і для альтернатив, виключних ситуацій (або одна для всіх). Поки післяумову не вказано в кожному конкретному потоці, вона має бути однаковою для всіх альтернативних потоків, а не тільки для основного.

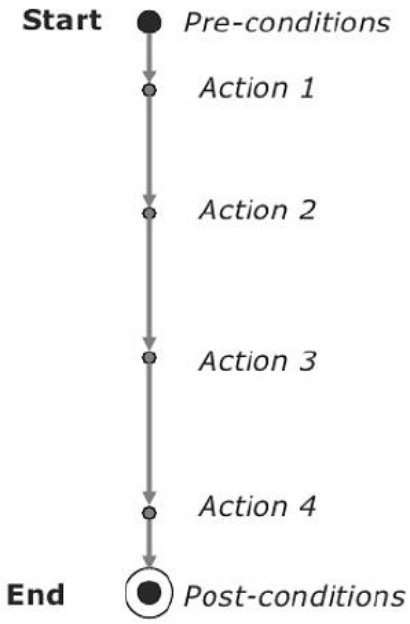


Рис. 1. Приклад основного потоку без альтернатив

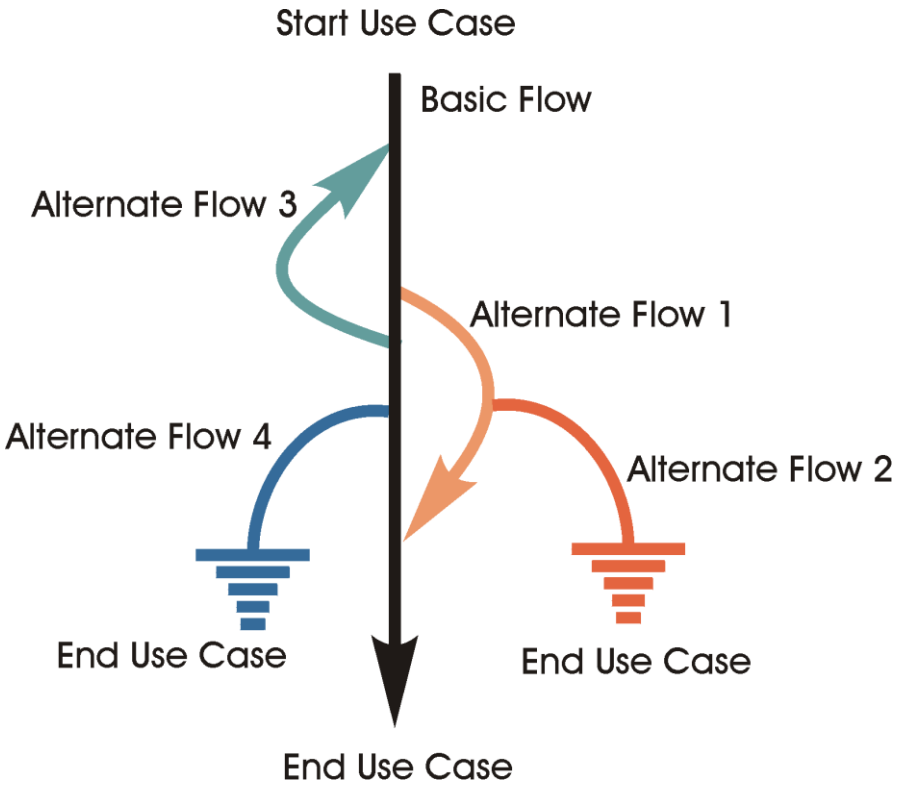


Рис. 2. Типи альтернативних потоків

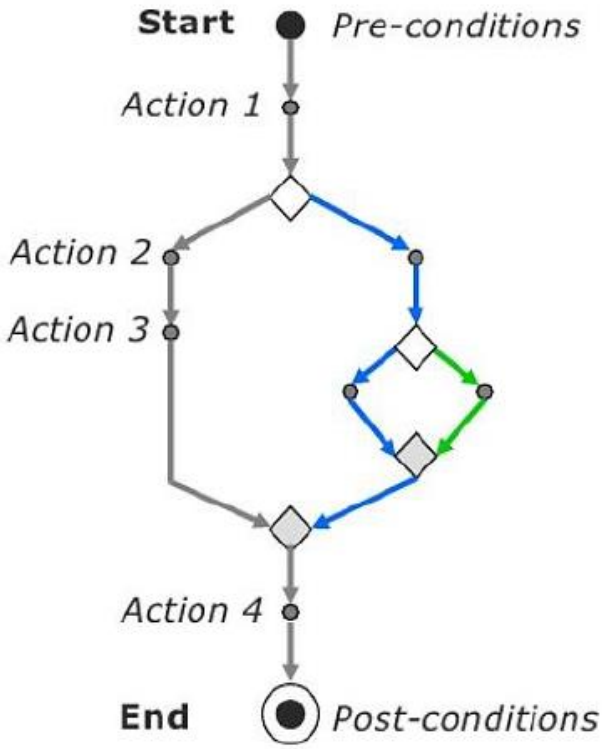


Рис. 3. Приклад ВВ з альтернативами, які не закінчують його виконання

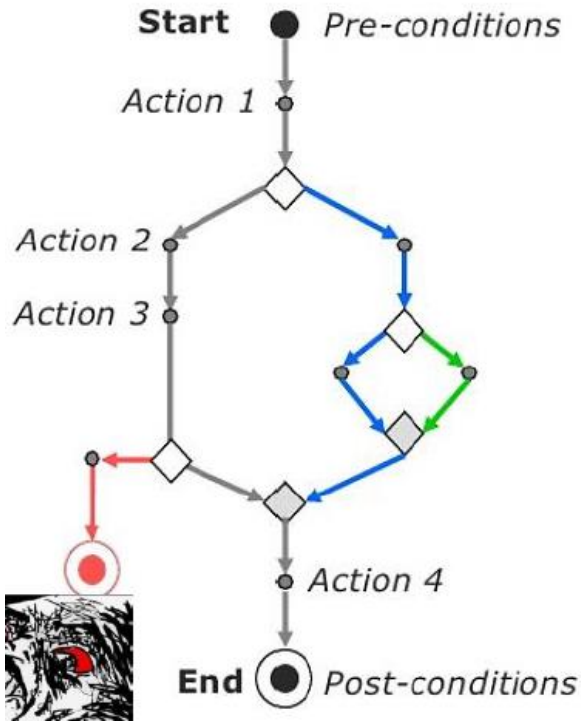


Рис. 4. Приклад ВВ який допускає аварійне завершення

Альтернативні потоки часто не повертаються в основний потік ВВ. Це пов'язано з тим, що вони зазвичай обробляють помилки і виключення основного потоку і мають інші післяумови.

Виключні ситуації пов’язані із потоком подій, які описують проблеми в самій системі. Вони теж не повертаються в основний потік ВВ.

Розширення – це внесення додаткового потоку подій до основного сценарію варіантів використання, настання якого залежить від певної умови, що виникає в точці розширення. Розширення можуть повертатися до кроку основного сценарію, з якого вони були викликані (внутрішня точка розширення), або система переходить до виконання іншого варіанту використання (зовнішня точка розширення).

Повнота функціональних вимог до розроблюваної системи досягається шляхом специфікації всіх варіантів використання з відповідними сценаріями, що відображають всі побажання і потреби користувачів системи.

Атрибути вимог

Атрибути вимог використовуються для надання їм певної характеристики та керування ними за цими ознаками.

В табл. 1 наведено атрибути, які використовуються в RUP.

Таблиця 1

Атрибути вимог за RUP

Атрибут	**Семантика**
Priority	Визначає пріоритет вимоги для призначення певних ресурсів розробки. High (Високий) Medium (Середній) Low (Низький)
Difficulty	Наскільки складна реалізація цієї вимоги.

	High (Висока), Medium (Середня) і Low (Низька).
Type	Functional (Функціональне) Usability (Зручність використання) Reliability (Надійність) Performance (Продуктивність) Supportability (Можливість супроводу) Design Constraint (Обмеження на Дизайн) Implementation Requirement (Вимога Реалізації) Physical Requirement (вимоги до апаратного забезпечення) Interface Requirement (Вимога Інтерфейсу)
Status	Відстежує прогрес розробки вимоги від початкової стадії до фінальної перевірки. Proposed (запропоновані) – функціональні особливості, які находяться в стані обговорення й ще не затверджені. Approved (схвалені) – функціональні особливості, затверджені для реалізації. Incorporated (включені) – функціональні особливості, які були реалізовані в певній версії. Validated (перевірені) – функціональні особливості, протестовані і перевірені на правильність роботи.
Benefit	Корисність і важливість вимоги для кінцевих користувачів і замовників. Critical (критичне) – вимога повинна бути реалізована, у противному випадку система не буде прийнята зацікавленими сторонами. Important (важливе) – вимога може бути опущена, але це несприятливо відіб'ється на зручності використання системи й задоволенні зацікавлених сторін.

Продовження табл. 1

	Useful (корисне) – вимога може бути опущена без істотного впливу на прийнятність системи.
Risk	Ризик, пов'язаний з додаванням цієї можливості: High (високий) – Серйозні наслідки ризику разом з високою можливістю його виникнення Medium (середній) – Наслідки ризику менш критичні, а можливість виникнення менше Low (низький). – Наслідки ризику мінімальні, а можливість виникнення ризику мала.
Effort	Оцінка часу й ресурсів, необхідних для реалізації можливості, виражена в людино-годинах.

1.2. Аналіз основних шаблонів текстових специфікацій ВВ

Ефективно специфікувати ВВ – описати його на такому рівні деталізації, який є достатнім для однозначного розуміння його сутності учасниками розробки і створення відповідної функціональності. При цьому витрати зусиль повинні бути мінімальними.

На сьогодні існують наступні загальноприйняті підходи до специфікації ВВ:

- чітко структурована документація на природній мові (текстова специфікація);
- графічні моделі, які описують варіанти використання у вигляді діаграм (найчастіше UML);
- формальні специфікації, де опис варіантів використання подано математично точними, формальними мовами (наприклад OKKAM).

Для документування ВВ найчастіше використовуються текстові специфікації, оскільки вони є зрозумілими ширшому колу зацікавлених

осіб, а використання діаграм та формальних мов потребують певних знань та умінь щодо їх використання.

Можна виділити наступні основні шаблони специфікації варіантів використання з застосуванням природних мов:

- реєстр ВВ;
- вільний формат;
- представлення кроків сценарію варіантів використання в табличному вигляді;
- повний формат (за Коберном);
- стиль RUP.

Розглянемо детальніше ці шаблони та проаналізуємо їх з точки зору повноти опису ВВ.

1.2.1. Реєстр варіантів використання

Перед тим як приступити до процесу специфікації, потрібно виявити всі можливі варіанти використання та сформувати відповідний реєстр акторів та варіантів використання.

Реєстр вимог є артефактом проекту, який містить записи про вимоги до розроблюваної системи та їх атрибути і використовується для керування вимогами.

Реєстр варіантів використання має вигляд таблиці, яка містить обов’язкові та додаткові поля. Обов’язковими полями є наступні:

- код – шифр, за яким можна однозначно ідентифікувати відповідний ВВ;
- назва ВВ – коротка фраза в вигляді іменника чи дієслова, що відображає мету ВВ;
- ціль – більш детальний опис мети ВВ;
- основний актор (дійова особа) – особа, яка ініціює ВВ.

Для шифрування ВВ можна використовувати наскрізну нумерацію по всім ВВ (1 – *N*), наскрізну нумерацію із вказанням типу вимоги (наприклад,

$UC_1 – UC_N$ і т.д.), нумерацію із вказанням актора, до якого відноситься відповідний ВВ (наприклад, $S_1 – S_m$ – для студента, $T_1 – T_k$ – для викладача).

Актор є будь якою зовнішньою, стосовно системи, що моделюється, сутністю, яка взаємодіє з системою та використовує її певні функціональні можливості для вирішення своїх цілей чи завдань. Також актори використовуються для позначення погодженої множини ролей, які можуть грати користувачі при взаємодії з системою, що проектується. Кожного актора можна розглядати як певну окрему роль відносно конкретного варіанта використання. Одна й та сама особа може виступати в декількох ролях і, відповідно, звертатися до різних сервісів системи.

Одним із акторів системи може виступати Таймер. Його вводять для того, щоб описати ВВ, які стосуються сервісів системи, що виконуються в певний момент часу (за розкладом).

Додатковими полями в реєстрі ВВ можуть виступати їх атрибути, які використовуються для керування ВВ. В табл. 2 наведено перелік атрибутів вимог, які рекомендовані для використання за методологією RUP.

Таблиця 2

Приклад реєстру варіантів використання

UC#	**Актор**	**Назва**	**Ціль**
U1	Користувач (Студент, Викладач)	Читання учбового матеріалу	Вказані користувачі мають змогу переглядати/читати учбові матеріали, що знаходяться в системі

Як бачимо, кількість інформації, яка міститься в реєстрі, є мінімальною.

1.2.2. Вільний формат

Реєстр ВВ містить список ВВ з якого можна дізнатися, який саме актор та для чого буде взаємодіяти із системою.

Вільний формат специфікації ВВ передбачає опис основного сценарію (дій актора і системи) в розповідному стилі. Вільний стиль допускає використання конструкцій «Якщо то».

Структура специфікації ВВ для цього випадку має такий вигляд:

1. Назва ВВ;
2. Основний актор
3. Ціль
4. Короткий опис основного потоку у довільній текстовій формі.

Приклад специфікації ВВ за таким шаблоном наведено нижче.

S1. Читання учбового матеріалу

Основний актор: користувач (студент, викладач)

Ціль: користувач має змогу переглядати/читати учбові матеріали, що знаходяться в системі

Основний потік: користувач переглядає список матеріалів. При виборі певного матеріалу система відображає на екрані його вміст. Якщо користувач не має доступу до певного матеріалу, він не відображається у списку.

Як бачимо, кількість інформації, яка міститься в такій специфікації, дозволяє описати успішних сценарій досягнення цілі актора.

1.2.3. Повний шаблон опису ВВ (за А. Коберном)

Повний шаблон опису варіанту використання за А. Коберном складається з наступних полів:

1. Назва (див. реєстр ВВ).
2. Контекст використання – деталізація мети та при необхідності опис умови нормального завершення.

3. Зона дії – опис рамок проекту. Наприклад – «Веб-ресурс для збору та аналізу статистичних даних про книжкові фонди загальноосвітніх навчальних закладів».
4. Основна дійова особа – (див. реєстр ВВ).
5. Учасники та інтереси – список інших учасників системи з вказанням їх інтересів.
6. Передумова – умова, виконання якої очікується перед ініціацією варіанту використання.
7. Мінімальні гарантії – те, що гарантується учасникам. Наприклад – у разі помилки дані відкочуються до попередньої версії.
8. Тригер – це те, що ініціює варіант використання, здебільшого це певна подія в системі.
9. Основний сценарій – в ньому описується основний сценарій в покроковому вигляді.

   Формат опису:

   Номер кроку

   Опис дії.
10. Розширення – послідовний опис всіх альтернативних сценаріїв, причому кожна з альтернатив прив'язується до певного кроку основного сценарію.

    Формат опису:

    Номер кроку

    Номер розширення

    Умова

    Дія або посилання на інший варіант використання чи альтернативу. Розширення закінчується описом умови виходу. Основні варіанти виходу з розширення це повернення основного сценарію в місце розгалуження, перехід до іншого кроку основного сценарію чи закінчення ВВ.

11. Список змін в технології і даних – що гарантується акторам-учасникам. Наприклад – в разі невдалої транзакції всі дані, що були в системі до її початку, зберігаються незмінними.
12. Допоміжна інформація – інформація, що додатково описує варіант використання.

Як бачимо, кількість інформації, яка міститься в специфікації за Коберном, дозволяє описати всі сценарії взаємодії актора із системою, а також додаткову інформацію.

1.2.4. Шаблон варіанту використання за методологією RUP

Методологія RUP пропонує власний шаблон текстової специфікації ВВ. Він містить наступні поля:

1. Назва та короткий опис.

   У даному розділі знаходиться: назва варіанту використання (див. реєстр), дійові особи (основна – з реєстру ВВ та вторинні), короткий опис варіанту використання (ціль із реєстру ВВ).
2. Основний потік.

   Тут описується основний сценарій у покроковому вигляді, починаючи від передумови і закінчуючи післяумовою. Кожний альтернативний сценарій виноситься в окремий параграф, та описується в стилі основного потоку подій. Альтернативні сценарії використовуються при відгалуженнях чи помилкових діях користувача ситуаціях та описують поведінку при них.
3. Спеціальні вимоги.

   Тут надається перелік нефункціональних вимог, що мають пряме відношення до обраного варіанту використання.
4. Передумови.

   Певні події які мають статися до ініціювання варіанту використання та необхідні для його виконання.
5. Постумови.

Опис очікуваних результатів виконання сценарію варіанту використання.

6. Точки розширення.

Визначення точок розширення та опис умов переходу за ними.

На рис. наведено фрагмент шаблону специфікації ВВ за RUP.

Як бачимо, кількість інформації, яка міститься в специфікації за RUP, дозволяє описати всі сценарії взаємодії актора із системою, а також містить іншу важливу інформацію.

1.2.5. Представлення ВВ в табличному вигляді

Інколи для специфікації основних сценаріїв варіантів використання використовують табличне представлення основного сценарію, приклади яких наведено нижче.

Таблиця 3

Приклад оформлення у вигляді таблиці в дві колонки

Актор	**Дія**
Користувач	Формує запит на пошук замовленнь
Система	Відображає список замовленнь
Користувач	Вибирає необхідне замовлення
Система	Показує докладну інформацію за замовленням

Таблиця 4

Приклад оформлення у вигляді таблиці в три колонки

№ кроку	**Користувач**	**Система**
1	Робить запит на пошук замовлень	Відображає список замовлень
2	Вибирає необхідне замовлення	Показує докладну інформацію за замовленням

Його перевагою є те, що при написанні специфікації чітко видно дії акторів та системи. Але така специфікація є малоінформативною, оскільки містить лише опис кроків основного сценарію.

1.3. Аналіз основних підходів до визначення рівня деталізації ВВ

Як бачимо із огляду шаблонів специфікацій ВВ, вони на різному рівні деталізації описують варіанти використання.

А.Коберном [1] запропоновано певні правила, щодо вибору шаблону та ступеню детальності текстової специфікації. Зусилля, витрачені на опис варіантів використання, він поділяє на чотири стадії точності по мірі зростання деталізації специфікації.

1. Дійові особи та цілі.
2. Короткий виклад варіанту використання або основний сценарій у формі кроків.
3. Список умов відмови – перелік ситуацій, які не входять до основного сценарію.
4. Обробка відмов – опис, як система повинна реагувати на кожну відмову.

В [2] на основі атрибутів Priority та Difficulty за RUP було запропоновано процедуру організації ефективної специфікації ВВ, яку можна використати при розроблені ПЗ.

Вона складається із наступних кроків:

1. Виявити всі можливі ВВ, які прийняті для реалізації в ПЗ.
2. Оформити виявлені варіанти використання у Реєстр ВВ
3. Виставити атрибути Priority та Difficulty для кожного ВВ.
4. На основі встановлених атрибутів визначити рівень специфікації кожного ВВ:

- при $P = L$ та $D = L$ обрати перший рівень деталізації – запис у реєстрі ВВ;

- при *P&D = H&L* або *M&M* обрати другий рівень деталізації, використавши шаблон вільного формату;
- при *P&D = H&M* або *M&H* обрати третій рівень деталізації, використавши шаблони Коберна або RUP без детального опису потоків обробки відмов;
- при *P&D = H&H*, необхідно специфікувати ВВ у повному об'ємі з деталізацією альтернатив, виключень та розширень.

1.4. Огляд ПЗ для специфікації ВВ

Для керування вимогами використовуються складні системи, які орієнтовані на роботу із вимогами, як елементами бази даних. Вимоги при цьому мають певний набір атрибутів, які використовуються для керування ними. За звичай документування варіантів використання у таких системах передбачає генерування певного шаблону специфікацій, які відповідають обраній методології. Так відомий комерційний продукт Rational Requisite PRO базується на методології RUP і, відповідно, використовує її шаблон та набір атрибутів.

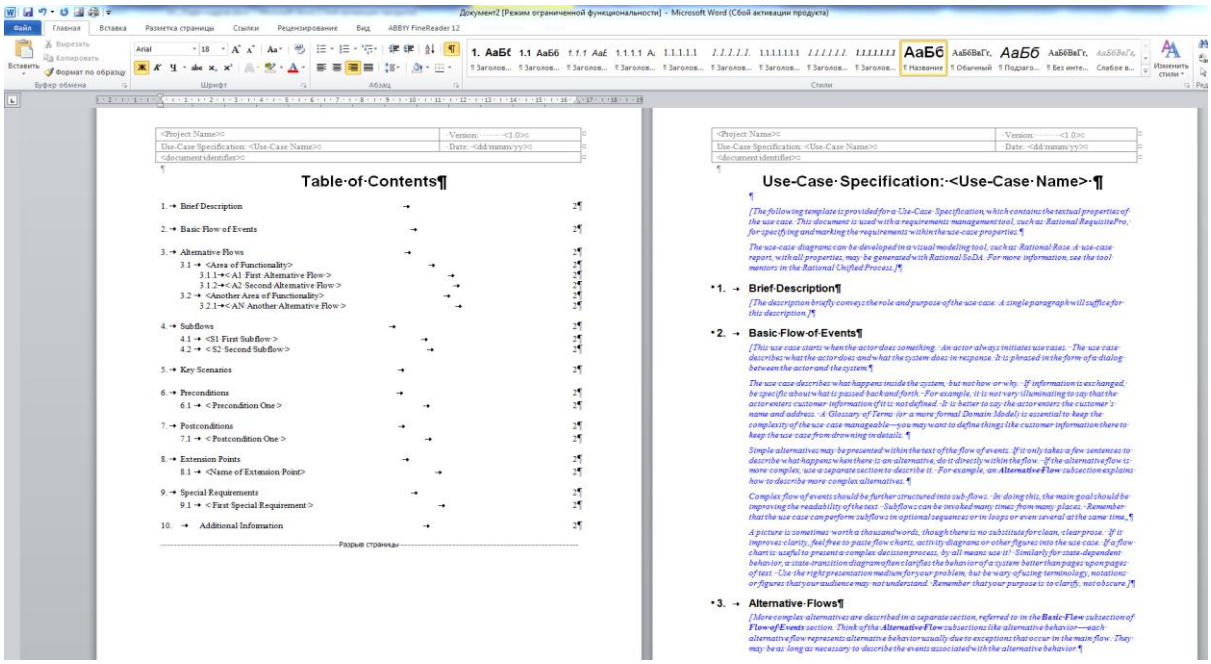


Рис. 5. Приклад шаблону для генерації специфікацій

1.5. Висновки до розділу 1

В даному розділі було проведено огляд та аналіз існуючих шаблонів текстових специфікацій варіантів використання. Показано, що ці шаблони на різних рівнях деталізації описують ВВ. Сучасні програмні засоби, які використовуються для роботи над проектами, пропонують використання певного шаблону, який є прийнятим для реалізації в цьому ПЗ і не надає можливість використання інших шаблонів, або зміни рівня деталізації відповідної специфікації.

2. СПОСІБ АВТОМАТИЗАЦІЇ ПРОЦЕСУ СПЕЦИФІКАЦІЇ ВАРІАНТІВ ВИКОРИСТАННЯ

2.1. Узагальнений метод визначення ступення деталізації специфікації ВВ на основі атрибутів

Метод визначення ступеня деталізації специфікації ВВ на основі атрибутів за RUP, описаний у пункті 1.4 має наступні обмеження:

- розрахований виключно на атрибути RUP;
- розрахований на використання двох атрибутів;
- є емпіричним (не має чіткої формули).

Отже, є необхідність у формулюванні узагальненого методу, який не матиме таких обмежень. Для цього необхідно встановити, що можна використовувати в якості атрибутів ВВ та які важливі характеристики мають ці атрибути.

2.1.1. Огляд атрибутів та їх характеристики

Очевидно, що із всього набору атрибутів, які пропонує RUP, найважливішим для прийняття рішення про ступінь деталізації є пріоритет ВВ (Priority). Він визначає пріоритет реалізації ВВ. При призначенні пріоритету вимозі враховуються її вплив на задоволеність користувача від реалізації цієї вимоги, частота та масовість використання ВВ. Існують і інші підходи до визначення пріоритету ВВ, аналіз яких дозволив сформулювати вимоги атрибутів, які можуть бути використані в узагальненому методі.

Метод оцінки пріоритету MoSCoW

При оцінці пріоритету методом MoSCoW для кожного завдання ми підбираємо найбільш відповідне йому 1 твердження з 4. Кожне із тверджень є індикатором певного рівня важливості завдання. Передбачає наявність таких груп пріорітетів:

- Must have this (це обов'язково має бути). «Must» вимоги не підлягають обговоренню і в будь-якому випадку повинні бути

реалізовані. Якщо вони не поставлені користувачеві в найближчих релізах, тоді весь проект не має сенсу.

- Should have this if at all possible (слід зробити, якщо це можливо). Дані завдання також важливі для користувача або замовника, проте, терміни їх поставки не так критичні як для «Must». Без цієї функціональності продукт вже можна використовувати, її відсутність не блокує базових можливостей зі списку «Must» і вона може бути видана трохи пізніше.
- Could have this if it does not affect anything else (можна зробити, якщо це не вплине на щось інше). Ця категорія завдань допускає максимально гнучкий погляд на такі основні критерії виконання як скоуп, терміни, якість і ресурси. Тут можлива варіативність і компроміси.
- Will not have this time but would like in the future (зараз на це немає часу, але хотілося б реалізувати в майбутньому). Для цієї категорії ключовим моментом є те, що дані вимоги можуть бути також важливі, також як і «Must», проте, їх можна залишити для більш пізніх поставок продукту.

Категорія «Will not» має 3 важливих ефекти:

1. Користувачам і замовникам не потрібно боротися, щоб включити будь-яку вимогу в Product Backlog.
2. Маючи можливість бачити список вимог для віддалених релізів, можна оцінити як це буде впливати на те, що буде реалізовано в першу чергу.
3. На вимоги з категорії «Will not» можна орієнтуватися як на довгострокову тенденцію в розвитку продукту, враховуючи дані особливості в ранніх релізах.

Після виконання таких дій кожна вимога матиме один з чотирьох пріоритетів.

Модель Кано для оцінки пріоритету

У цьому методі вимоги класифікуються на основі переваг клієнта по 5 категоріям. Кожна категорія показує наскільки важлива користувачеві, замовнику або клієнту та чи інша функціональність, якість або характеристику продукту:

- Must-be Quality – обов'язкові для користувача характеристики продукту. Це вимоги, які замовники, користувачі або клієнти очікують в першу чергу і сприймаються як належне, вони є необхідними для випуску продукту. Коли ці функції реалізовані, клієнти сприймають це як щось стандартне тобто ті, що мають бути за замовчуванням, але коли вони не реалізовані або зроблені погано, клієнти дуже незадоволені.
- One-dimensional Quality – одномірні характеристики. Це функціональність, яка призводить до задоволення користувачів або клієнтів, коли вона реалізована і невдоволення, коли її немає. Це параметри і можливості продукту, на яких будується конкуренція компаній між собою.
- Attractive Quality – привабливі для користувача характеристики. Реалізація цих вимог забезпечує задоволення користувачів, але при цьому коли вони не виконуються, це не викликає незадоволення. Це функціональність, яка зазвичай не очікується і радує користувача, якщо вона раптом з'явилась.
- Indifferent Quality – байдужі для користувача характеристики. Для користувача вимоги з цієї категорії не є ні добрими, ні поганими, вони не приводять до задоволення або незадоволення клієнтів. Прикладом таких завдань можуть бути різні технічні рішення, які приховані від користувача.
- Reverse Quality – протилежні характеристики. До цієї групи належать вимоги, реалізація яких покращує, але при цьому ускладнює продукт. Це можуть бути різні додаткові функції, які

розширюють необхідну і достатню функціональність продукту. Для ряду клієнтів і користувачів складний високотехнологічний продукт, з великою кількістю функцій і можливостей може бути надто складним для використання.

Дана модель дозволяє характеризувати кожну задачу, вибудувати черговість реалізації вимог і сформувати план реалізації функціональності базуючись на очікуваннях користувачів, керуючи їх реакцією на висновок тієї або іншої функціональності.

В цьому методі дій кожна вимога має один з п’яти пріоритетів.

Пріоритет та інші атрибути ВВ за RUP

Атрибути RUP, в своїй більшості, мають ступінь градації рівний трьом, а саме «Високий», «Середній» та «Низький».

Виключенням є атрибут «Тип», оскільки він показує те, до якого типу відносяться вимоги. Тобто даний атрибут не доцільно розглядати в контексті поставленої задачі, оскільки він не дає кількісних характеристик та не може бути застосований при виборі рівня деталізації.

Також доцільно пропустити атрибут «Статус», оскільки він також не має кількісного представлення та використовується для того, щоб показати те, до якої з груп відноситься вимога, а саме: «Запропоновані» – функціональні особливості, які находяться в стані обговорення й ще не затверджені; «Схвалені» – функціональні особливості, затверджені для реалізації; «Включені» – функціональні особливості, які були реалізовані в певній версії; «Перевірені» – функціональні особливості, протестовані і перевірені на правильність роботи.

Також існує атрибут «Трудомісткість», який має чисельне представлення в людино-годинах, але не має визначеного ступеню градації. Його можна застосувати у тому випадку, якщо розробити співвідношення для кожного рівня, обраної для нього градації, та значення в людино-годинах, або просто використовувати як приблизну характеристику з заданим ступенем градації.

Тепер розглянемо список атрибутів, які без правок придатні для вибору рівня деталізації:

- «Пріоритет» – визначає пріоритет вимоги для призначення певних ресурсів розробки
- «Складність» – визначає складність розуміння вимоги та технічну складність її реалізації
- «Корисність» – визначає корисність ті важливість вимоги для кінцевих користувачів та замовників
- «Ризик» – визначає ризик, пов'язаний з реалізацією вимоги

Дані атрибути мають три рівні градації та можуть використовуватися для вибору рівня деталізації.

2.1.2. Узагальнений метод визначення ступеня деталізації специфікації ВВ

Отже, пропонується в якості вимоги до атрибутів, обраних користувачем, встановити ступінь градації їх значень від 3 до 5, оскільки це надасть сумісність з іншими методиками пріоретизації, такими як MoSCoW, Кано, RUP та ін. Для інших методів, в яких використовуються атрибути, але їх ступені градації більші за 5, можна застосувати приведення до одного з допустимих ступенів.

Очевидно, що кожен атрибут повинен мати індивідуальний вплив на ступінь деталізації специфікації ВВ. Тому узагальнена формула повинна містити вагові коефіцієнти цих атрибутів.

Отже, на основі вищевказаних вимог запропоновано формулу вибору рівня деталізації, яка базується на основі середньої арифметичної. Її математичне представлення наведено нижче:

$$\mathrm{f}(A_1, \dots, A_n) = \left\lfloor \frac{\sum(C_i * A_i * \frac{4}{Cg_i})}{\sum C_i} \right\rfloor,$$

де Cg_i – ступінь градації атрибутів

A_i – значення атрибутів ($1 \dots Cg_i$)

C_i – значення відповідних вагових коефіцієнтів атрибутів

Частина $\frac{\sum(C_i * A_i)}{\sum C_i}$ реалізує вплив значеннь атрибутів та відповідних вагових коефіцієнтів на вибір рівня деталізації. Тобто з допомогою вагових коефіцієнтів можна регулювати значущість атрибутів при виборі рівня деталізації, причому немає обмежень на діапазон значень цих коефіцієнтів, крім того, що вони мають бути дійсними та строго додатними, що дозволяє більш зручно налаштовувати вплив атрибутів. Проте, при надто великій різниці між коефіцієнтами може статися така ситуація, коли вплив на вибір чинить лише один або кілька атрибутів, коли інші практично повністю втрачають значимість. В таких ситуаціях потрібно переглянути самі атрибути, та прибрати чи замінити ті, які чинять надто малий вплив на вибір рівня деталізації чи переглянути встановлені вагові коефіцієнти. Наприклад, коли наявний атрибут з найвищим коефіцієнтом впливу, наприклад 1, не має сенсу вводити атрибути у яких їх коефіцієнт впливу буде меншим за 0.3, оскільки це призведе до зайвої роботи з ними, оскільки вони практично не будуть ні на що впливати.

Частина $\frac{\sum(C_i * A_i)}{\sum C_i}$ з округленням до найближчого цілого може бути самостійно використана для вибору рівня деталізації, але тоді і тільки тоді, коли ступінь градації атрибутів дорівнює 4, оскільки ми маємо 4 рівня деталізації. Але основні методики пріоретизації мають ступені градації від 3 до 5, або можуть бути приведені до таких, тому було введено частину $\frac{4}{Cg_i}$ яка використовується для приведення отриманих значень до діапазону від 1 до 4.

Проте існують 2 дефекти при такому перетворенні. Коли ступінь градації дорівнює 3 – ми маємо мінімально можливе значення 1.3(3), що не суттєво ускладнює потрапляння в першу категорію та переважує значення до більш детальних категорій. При ступені градації 5 – ми маємо мінімально можливе значення 0.8, що не суттєво полегшує потрапляння в першу

категорію та переважує значення до менш детальних категорій. Характерною особливість таких дефектів є спадання до нуля на протязі всього діапазону рівнів деталізації, та в районі четвертої категорії, прямує до нуля.

Попри це, дані дефекти не є суттєвими, оскільки задача має низьку чутливість до похибок (через округлення до найближчого цілого), а також розподіл варіантів використання підпорядковується нормальному розподілу, причому на першу та четверту категорію припадає приблизно по 10%, решта - на другу та третю. Тобто маємо невелику ймовірність внесення похибки в її максимальному вигляді, оскільки похибка спадає при підході до більш детальних категорій та повністю зникає в районі четвертої. Крім цього, незалежно від дефекту, при гранично низьких значеннях гарантується потрапляння в першу категорію, а при гранично високих – у четверту. Також при потребі, частина за приведенням до значеннь від 1 до 4 може бути легко замінена на іншу аналогічну чи модифікована, оскільки вона тісно не зв’язана з основною формулою, а є лише її множником, перед округленням до найближчого цілого. Як приклад можна розглянути приклад формули з усуненням даних дефектів:

$$\mathrm{f}(A_1, \dots, A_n) = \left\lfloor \frac{\sum(C_i * g(A_i, Cg_i))}{\sum C_i} \right\rceil$$

$$g(A_i, Cg_i) = \begin{cases} 1, & A_i < (3+k)/2 \\ 2, & (3+k)/2 \le A_i < (5+3*k)/2 \\ 3, & (5+3*k)/2 \le A_i \le (7+5*k)/2 \\ 4, & A_i > (7+5*k)/2 \end{cases}$$

$$k = \frac{Cg_i - 4}{3},$$

де Cg_i – ступінь градації атрибутів

A_i – значення атрибутів $(1 \dots Cg_i)$

C_i – значення відповідних вагових коефіцієнтів атрибутів

Даний варіант формули вибору рівня деталізації має більшу точність, ніж спрощена формула, яка описана вище. Проте дефекти приведення, першої формули є не настільки суттєвими, щоб за них розплачуватись

підвищенням складності обчислень та програмної реалізації. Крім того, не обов'язково всі атрибути матимуть один ступінь градації, тобто похибка може бути майже повністю скомпенсована в ситуації коли атрибути мають різні ступені градації, тобто похибка внесена атрибутами з ступенем градації рівним 3, частково компенсується атрибутами з ступенем градації рівним 5. З цього виходить те, що описані значення похибки справедиві лише для ситуації коли атрибути мають один ступінь градації (3 чи 5) та виборі мінімальних значень атрибутів, а при різних ступенях вплив цих похибок ще суттєво зменшується. Тому рекомендовано використовувати спрощену версію.

Відмітимо, що використання великої кількості атрибутів ВВ є недоцільним, оскільки при такій ситуації складність вибору рівня деталізації стане рівною чи більшою ніж вибір вручну методом експертної оцінки.

Приклад

Для автоматизованої системи по роботі з рахунками клієнтів визначено набір з 8 основних варіантів використання, спрощений реєстр яких подано в табл. 5.

Таблиця 5

Спрощений реєстр ВВ

№	Назва ВВ
UC1	Друкувати інформацію про операції
UC2	Друкувати інформацію про рахунок
UC3	Змінити баланс рахунку
UC4	Нарахувати відсотки
UC5	Переглянути список клієнтів та їх рахунки
UC6	Редагувати дані по клієнта (CRUDL)
UC7	Створити або видалити рахунок
UC8	Увійти в систему

Розглянемо приклад роботи узагальненого методу вибору рівня деталізації з двома атрибутами, дані про які подано в табл. 6.

Таблиця 6

Атрибути ВВ та їх характеристики

i	**Атрибут**	**Зміст атрибуту**	C_i	Cg_i
1	Priority (MoSCoW)	рівень зацікавленості дійових осіб у реалізації ВВ	0,7	4
2	Difficulty (RUP)	рівень складності реалізації	1	3

Відповідно до формули отримуємо значення рекомендованого рівня деталізації, які подано в табл. 7 (записи сортовано за зменшенням рівня деталізації).

Таблиця 7

Варіанти використання та їх характеристики

№	**Priority**	**Difficulty**	**Рекомендований рівень деталізації**
UC6	Must (4)	High (3)	4
UC3	Could (2)	High (3)	3
UC7	Must (4)	Medium (2)	3
UC4	Won't (1)	High (3)	3
UC5	Must (4)	Low (1)	2
UC1	Won't (1)	Medium (2)	2
UC8	Should (3)	Low (1)	2
UC2	Won't (1)	Low (1)	1

Зауважимо, що при достатньо великій вибірці, рекомендований рівень деталізації ВВ підпорядковується нормальному закону розподілу, причому на першу та четверту категорію припадає приблизно по 10%, решта на другу

та третю. Також це відповідає рекомендаціям по вибору рівня деталізації експертним шляхом для середньостатистичного проекту.

2.2. Вибір шаблонів специфікацій ВВ

2.2.1. Специфікація ВВ першого рівня деталізації

В процесі специфікації варіантів використання перед вибором рівнів деталізації та самим написанням специфікацій, проводиться відбір ВВ для реалізації та їх запис у реєстрі ВВ. Тобто не залежно від рівня деталізації, всі варіанти використання, які допущено до реалізації, повинні мати опис у реєстрі ВВ.

Для кожного рівня деталізації було обрано певний шаблон специфікації за виключенням першого рівня, оскільки при ньому написання специфікацій не проводиться, а варіант використання описується лише записом в реєстрі ВВ. Це дозволяє не витрачати час на деталізацію несуттєвих варіантів використання, але при цьому мати достатньо інформації про те, хто є ініціатором, тобто головною дійовою особою та про ціль виконання ВВ. Дані поля дають приблизне уявлення про результати виконання та дійову особу, але вони не дають інформації про умови виконання та інші нюанси реалізації, тому перший рівень деталізації не підходить для більш важливих чи складних варіантів використання. Для таких ВВ проводиться процедура написання специфікацій з застосуванням шаблонів, які містять деталі взаємодії актора та системи.

Таблиця 8

Приклад шаблону специфікації ВВ 1 рівня

UC#	Опис
Назва	Опис
Актор	Опис
Ціль	Опис

2.2.2. Специфікація ВВ другого рівня деталізації

На другому рівні деталізації дані в реєстрі ВВ розширюються коротким описом основного потоку варіанта використання у довільній текстовій формі. Даний шаблон є компромісом між записом у реєстрі ВВ та детальним описом.

Таблиця 9

Приклад шаблону специфікації ВВ 2 рівня

UC#	Опис	Інформація з 1 рівня
Назва	Опис	
Актор	Опис	
Ціль	Опис	
Основний потік	Опис у довільній текстовій формі	Розширення 2 рівня

2.2.3. Специфікація ВВ третього рівня деталізації

На третьому рівні деталізації замість короткого опису основного потоку в довільній текстовій формі вводиться опис основного потоку в покроковому вигляді.

Також з’являється короткий опис, який містить інформацію про варіант використання у дуже короткій формі. Приклад короткого опису, який використовується в шаблонах третього та четвертого рівня: «Даний варіант використання дозволяє менеджеру додавати, видаляти і редагувати дані клієнта.».

Крім цього на третьому рівні деталізації додаються умови, які поділяються на такі типи: передумова, післяумова, спеціальна умова.

На даному рівні деталізації з'являються тригери, які являються полями з інформацією про те, які саме події призводять до ініціювання виконання варіанту використання.

Додатково на цьому рівні деталізації надається список альтернатив та виключних ситуацій. Альтернативи являються відгалуженнями від основного потоку та можуть як повернутися до певного кроку основного потоку, після свого виконання, так і завершити варіант використання без повернення. Виключні ситуації описують поведінку при певних збоях системи при виконанні варіанта використання, при чому можуть як аварійно завершити виконання ВВ, так і перейти до певного кроку основного сценарію.

Даний шаблон дозволяє достатньо точно та однозначно зрозуміти поведінку актора та системи в межах варіанта використання та можливі виключні ситуації, які потребують опрацювання при реалізації.

Таблиця 10

Приклад шаблону специфікації ВВ 3 рівня

UC#	Опис	Інформація з 1 рівня
Назва	Опис	
Актор	Опис	
Ціль	Опис	
Тригер	Список	Розширення 2 рівня
Передумова	Список	
Основний потік	Покроковий опис	Заміна 2 рівня
Післяумова	Список	Розширення 2 рівня
Альтернативи	Список	
Виключні ситуації	Список	
Розширення	Список	
Спеціальні умови	Список	

2.2.4. Специфікація ВВ четвертого рівня деталізації

На четвертому рівні деталізації специфікацій ВВ зберігаються всі поля третього рівня.

В шаблон додається опис точок розширення, альтернативи та виключні ситуації мають власні покрокові описи потоків виконання, по типу опису основного потоку ВВ. Точки розширення бувають зовнішніми та внутрішніми. Зовнішні точки розширення описують варіант використання, який ініціюється, та умову спрацювання. Внутрішні точки розширення не ініціюють інших варіантів використання, а лише мають певний набір кроків та умову при яких вони спрацьовують. По своїй поведінці внутрішні точки розширення схожі на альтернативи, оскільки також можуть привести до завершення виконання варіанту використання та мають певні кроки. Проте, якщо при виконанні внутрішньої точки розширення не завершено варіант використання, то по закінченню її дії проходить повернення строго до того кроку основного потоку, де було ініційовано точку розширення та виконання основного потоку продовжується. Альтернативи, в свою чергу, після виконання можуть повернутись до довільного кроку основного потоку та, як і виключні ситуації, не виходять за рамки варіанту використання.

Даний шаблон дозволяє максимально точно та однозначно описати поведінку варіанта використання.

Таблиця 11

Приклад шаблону специфікації ВВ 4 рівня

UC#	Опис	Інформація з 3 рівня
Назва	Опис	
Актор	Опис	
Ціль	Опис	
Тригер	Список	
Передумова	Список	
Основний потік	Покроковий опис	
Післяумова	Список	
Альтернативи	Покроковий опис	Заміна 3 рівня

Продовження табл. 11

Виключні ситуації	Покроковий опис	
Розширення	Покроковий опис	
Спеціальні умови	Список	Інформація з 3 рівня

2.2.5. Модульність подання специфікації ВВ

Використання вказаних чотирьох рівнів шаблонів текстових специфікацій надає можливість зберігати готові специфікації у модульному вигляді та при потребі нарощувати їх деталізацію. Специфікація варіанту використання вже розглядається не як монолітний об'єкт, а як набір модулів, які значно легше супроводжувати.

Це дозволить використовувати інформативну частину специфікацій без жорсткої прив’язки до документування за рахунок структурованого подання інформації та відповідних посилань.

Модульний підхід дозволяє точно локалізувати зміни, без необхідності повного перегляду та порівняння з старою версією специфікації. На основі модульного подання також можна реалізувати генерацію специфікацій у класичному вигляді, з використанням заготовленого шаблону.

2.3. Процедура автоматизованого створеня специфікації ВВ

На основі описаного вище узагальненого методу визначення ступеня деталізації специфікації можливо частково автоматизувати процес створення специфікації варіантів використання, надаючи користувачеві рекомендації щодо вибору рівня специфікації ВВ та відображаючи відповідний шаблон.

Запропонований автоматизований процес створення специфікацій ВВ матиме такі кроки:

1. користувачем в системі створюється реєстр всіх зібраних варіантів використання, як при класичному підході без автоматизації;

2. користувачем обираються атрибути та, при потребі, змінюються їх вагові коефіцієнти;
3. система обчислює рекомендований рівень деталізації ВВ та надає відповідний шаблон специфікації;
4. користувач наповнює даними специфікацію варіанту використання. Якщо користувач вирішить використати інший рівень деталізації, відмінний від рекомендованого системою, система надасть користувачеві відповідний шаблон.

2.4. Висновки до розділу 2

У даному розділі було проведено аналіз атрибутів, які використовуються в різних методах пріоретизації вимог та розроблено метод вибору рівня деталізації специфікації варіанту використання. Метод MoSCoW та модель Кано в їх класичному вигляді є сумісними з розробленим алгоритмом, проте ефективніше використовувати власні атрибути чи комбінацію з атрибутів даних методів, оскільки це розбиває задачу визначення пріоритету на кілька простіших – обрати значення атрибутів.

Крім цього було обрано такі шаблони специфікацій, які дозволяють легко нарощувати рівень деталізації та спростити супровід за рахунок їх модульного подання.

Описана в цьому розділі процедура автоматизованого створення специфікації ВВ базується на сформульованому методі вибору рівня деталізації та використовує модульне подання специфікації.

3. АНАЛІЗ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СПОСОБУ

Розроблювальна система являє собою веб-додаток для автоматизації процесу специфікації варіантів використання на основі їх атрибутів. Його основною задачею є вибір рекомендованого рівня деталізації на основі атрибутів, заданих користувачем, надання зручного інтерфейсу, який автоматично адаптується під обраний рівень деталізації, що в свою чергу дозволяє полегшити процедуру написання та супроводу специфікацій. Також системою передбачено зручне нарощування рівня деталізації специфікацій.

3.1. Аналіз засобів розроблення програмного забезпечення

Python – потужна і проста для вивчення мова програмування. Вона дозволяє використовувати ефективні висококореневі структури даних та пропонує простий, але ефективний підхід до об'єктно-орієнтованого програмування. Суміш вдалого синтаксису, динамічна типізація в інтерпретаційній мові робить Python ідеальною мовою для написання сценаріїв та прискореного розроблення, застосунків у різних сферах та на більшості платформ. Інтерпретатор Python та його стандартна бібліотека, що постійно розширюється та перебуває у вільному доступі у вигляді сирцевого коду та бінарних файлів для всіх основних платформ на офіційному веб-сайті Python та можливість розповсюдження без обмежень. Крім цього на сайті зберігаються дистрибутиви та посилання на численні модулі від інших розробликів для мови Python, різноманітні програми та інструменти, а також додаткова документація. Інтерпретатор Python може бути легко розширений за допомогою нових функцій і типів даних, написаних на C/C ++.

Django – безкоштовний фреймворк для веб-застосунків на мові Python, що використовує MVC-шаблон проектування. Проект підтримується організацією Django Software Foundation. Сайт на Django будується з одного або кількох застосунків, які рекомендовано розробляти незалежними та з

можливістю підключення та відключення. Це одна з суттєвих архітектурних відмінностей цього фреймворку від деяких інших, таких як, Ruby on Rails. Один з основних принципів фреймворку – DRY, а саме «Не повторюйте себе». Також, на відміну від інших фреймворіків, обробники URL в Django конфігуруються в явному вигляді за допомогою регулярних виразів. Для роботи з базою даних Django використовує власний ORM, в якому модель даних описується класами Python, і по ній генерується схема бази даних.

MariaDB – це альтернативна MySQL СКБД, яку розробляє автор MySQL Michael "Monty" Widenius. Основна ціль проекту MariaDB – створення новністю бінарно сумісної з оригінальною MySQL версії СКБД, яка при цьому буде мати значну кількість покращенн в коді, що впливають на продуктивність. MariaDB розробляється як drop-in заміна для MySQL, що повністю імітує поведінку MySQL.

3.2. Архітектура розробленого програмного забезпечення

Даний веб-додаток можна глобально поділити на такі частини:

- Модель (model);
- Представлення (view);
- Шаблон (template).

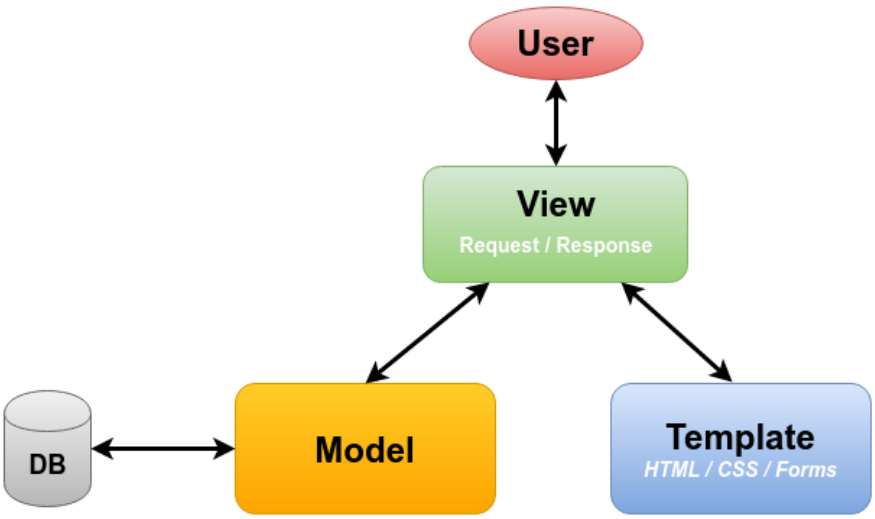


Рис. 5. Приклад шаблону MVT

Модель – на цьому рівні проводяться всі маніпуляції з даними та реалізуються функції, які готують потрібні дані для представленнь

Представлення – рівень який відображає бізнес-логіку системи. Він містить логіку про те, які дані потрібно отримати з моделі та який шаблон потрібно застосувати. Тобто фактично він виконує функції контроллера. Також на цьому рівні проводиться контроль доступу до можливостей системи.

Шаблон – шар представлення даних. Тут відбувається генерація сторінок, які бачать користувачі, тобто цей рівень відповідає за те, щоб інтерфейс адаптувався під тип користувача та під рівень деталізації, при роботі з специфікаціями.

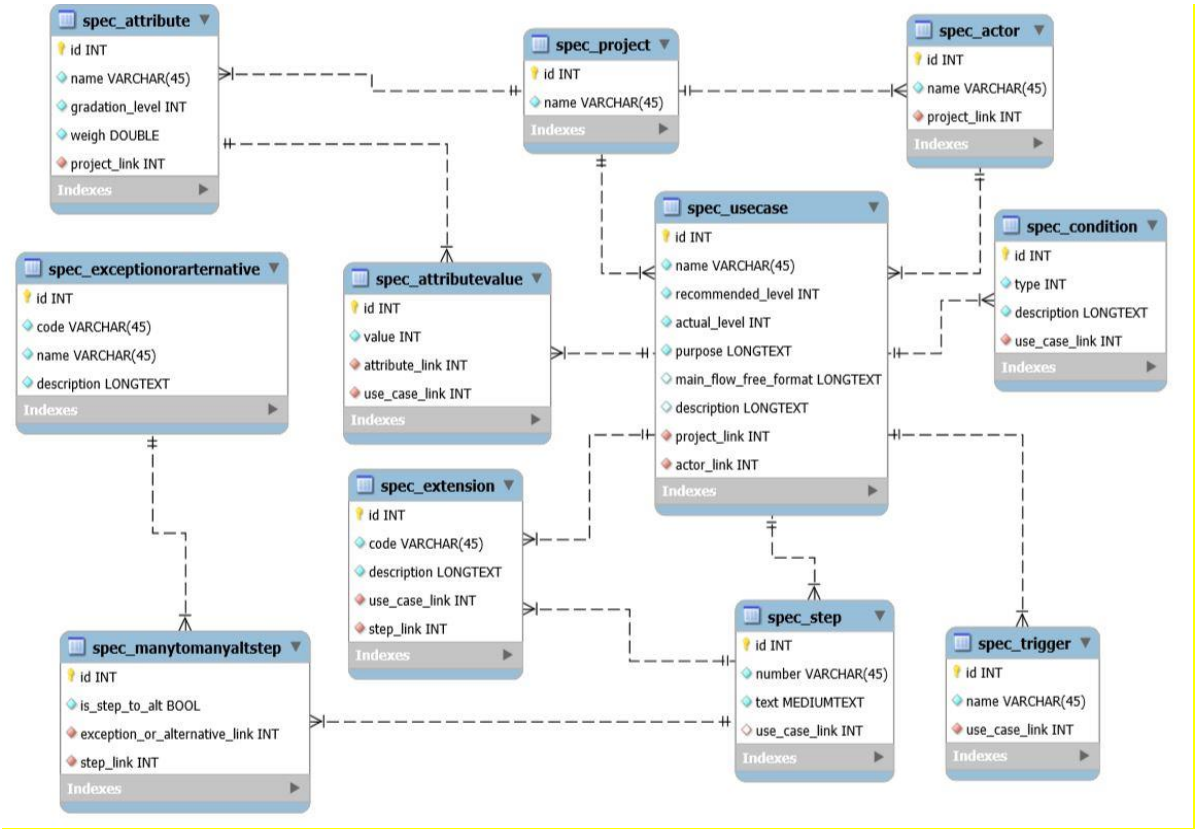


Рис. 6. ER діаграма застосунку

Лістинг 1. Клас сутності Project

```
class Project(models.Model):
    name = models.CharField(max_length=250)
```

Project – об’єднує між собою специфікації для конкретного проекту та має лише одне поле назви. Використовується для розмежування атрибутів, дійових осіб та варіантів використання з їх супутніми елементами, між кількома паралельними проектами та для зручного видалення даних, при необхідності.

Лістинг 2. Клас сутності Actor

```
class Actor(models.Model):
    name = models.CharField(max_length=250)
    project_link = models.ForeignKey(Project, on_delete=models.CASCADE)
```

Actor – використовується для зручності заповнення поля «Дійова особа» в специфікаціях та реєстрі ВВ, а саме надає список для вибору користувачеві, що в свою чергу мінімізує ручний ввід та помилки.

Лістинг 3. Клас сутності Attribute

```
class Attribute(models.Model):
    weigh = models.FloatField(default=1.0)
    name = models.CharField(max_length=250)
    gradation_level = models.IntegerField(choices=GRADATION_CHOICES,
default=4)
    project_link = models.ForeignKey(Project, on_delete=models.CASCADE)
```

Attribute — використовується при виборі рівня деталізації, а саме надає список атрибутів, їх ваги та ступені градації.

Лістинг 4. Клас сутності UseCase

```
class UseCase(models.Model):
    name = models.CharField(max_length=250)
    recommended_level = models.IntegerField(default=1)
    actual_level = models.IntegerField(default=1)
    project_link = models.ForeignKey(Project, on_delete=models.CASCADE)
    actor_link = models.ForeignKey(Actor, blank=True, null=True,
on_delete=models.SET_NULL)
    description = models.TextField(blank=True, null=True)
    purpose = models.TextField()
    main_flow_free_format = models.TextField(blank=True, null=True)
```

UseCase – сутність яка зберігає базову інформацію про варіант використання, а саме назву, рекомендований рівень деталізації, актуальний

рівень деталізації, посилання на головну дійову особу, яка є його ініціатором, короткий опис, короткий опис у довільній текстовій формі та ціль. Короткий опис та короткий опис основного потоку є не обов'язковими полями, та заповнюються при потребі, вдповідно до шаблону обраного рівня деталізації.

Лістинг 5. Клас сутності AttributeValue

```
class AttributeValue(models.Model):
    attribute_link = models.ForeignKey(Attribute, on_delete=models.CASCADE)
    use_case_link = models.ForeignKey(UseCase, on_delete=models.CASCADE)
    value = models.IntegerField(default=1)
```

AttributeValue – сутність яка зберігає значення атрибутів для варіантів використання. Створення або видалення атрибутів чи варіантів використання призводить до змін в записах даної сутності. При створенні нового атрибуту, додаються записи, відповідно всім ВВ які заповнюються мінімальним значенням з діапазону градації, а саме 1. При видаленні атрибуту видаляються всі записи, зовнішнім ключем яких є атрибут, який видаляється, точно так же як і з видаленням варіанта використання, тільки там буде видалено всі записи, зовнішнім ключем яких є ВВ, який видаляється. При додаванні нового варіанту використання, автоматично створюються записи, для кожного атрибута в проекті, на виставляє відповідні значення в 1. Тобто після створення, ВВ за замовчуванням має значення атрибутів рівними 1.

Лістинг 6. Клас сутності Condition

```
class Condition(models.Model):
    type = models.IntegerField(choices=TYPE_CHOICES, default=1)
    description = models.TextField()
    use_case_link = models.ForeignKey(UseCase, on_delete=models.CASCADE)
```

Condition – сутність яка зберігає множини умов варіантів використання в проекті. Поділяється три типи: передумови, післяумови та спеціальні умови. Крім типу має опис, в якому відображається суть умови.

Лістинг 7. Клас сутності Step

```
class Step(models.Model):
    number = models.CharField(max_length=20)
    text = models.TextField()
    use_case_link = models.ForeignKey(UseCase, blank=True, null=True,
on_delete=models.CASCADE)
```

Step – сутність яка зберігає множини кроків варіантів використання, альтернатив та виключних ситуацій. Має поля номер, текст кроку та не обов'язковий зв'язок з варіантом використання, який є лише тоді, коли крок відноситься до ВВ. Коли крок вдноситься до альтернативи чи виключної ситуації, даний зовнішній ключ пустий, проте є відповідні записи у сутності ManyToManyAltStep.

Лістинг 8. Клас сутності ManyToManyAltStep

```
class ManyToManyAltStep(models.Model):
    step_link = models.ForeignKey(Step, on_delete=models.CASCADE)
    exception_or_alternative_link =
models.ForeignKey(ExceptionOrAlternative, on_delete=models.CASCADE)
    is_step_to_alt = models.BooleanField()
```

ManyToManyAltStep – технічна сутність яка зберігає дані про відношення між кроками і альтернативами та виключними ситуаціями. Також має інформацію про напрямок зв’язків, що використовується для ідентифікації батьківського кроку альтернативи та батьківської альтернативи кроку. Це реалізовано тому, що в розробленій системі допускаються вкладені альтернативи та виключні ситуації, причому рівень вкладеності може бути довільним.

Лістинг 9. Клас сутності ExceptionOrAlternative

```
class ExceptionOrAlternative(models.Model):
    code = models.CharField(max_length=50)
    description = models.TextField()
    name = models.CharField(max_length=250)
```

ExceptionOrAlternative – сутність яка відображає альтернативи та виключні ситуації. Між собою вони відрізняються лише кодами, тобто виключні ситуації починаються з букви «E», наприклад «E1», а альтернативи з букви «A», наприклад «A2.4». Записи крім коду мають назви та умови спрацювання.

Лістинг 10. Клас сутності Extension

```
class Extension(models.Model):
    step_link = models.ForeignKey(Step, on_delete=models.CASCADE)
    use_case_link = models.ForeignKey(UseCase, on_delete=models.CASCADE)
    description = models.TextField()
    code = models.CharField(max_length=50)
```

Extension – сутність яка відображає точки розширення. Записи мають поля код та опис. Також кожен запис має зв’язок по зовнішньому ключу з кроком, на якому спрацьовує точка розширення, та з варіантом використання, який ініціюється при її спрацюванні.

Лістинг 11. Клас сутності Trigger

```
class Trigger(models.Model):
    name = models.CharField(max_length=250)
    use_case_link = models.ForeignKey(UseCase, on_delete=models.CASCADE)
```

Trigger – сутність яка відображає множину тригерів для варіанта використання. Не включена в поля сутності UseCase тому, що ВВ може мати більше одного тригера.

3.3. Особливості реалізації програмного застосунку

3.3.1 Аналіз основних типів функцій рівня моделі

Лістинг 12. Приклад типової функції вибірки об’єкта

```
def get_use_case_by_id(use_case_id):
    return UseCase.objects.get(id=use_case_id)
```

Функції даного типу використовуються в якості зручних обгорток над синтаксисом Django-models. Використовуються лише в тих випадках, коли точно відомо що шуканий об’єкт існує та єдиний, оскільки в інших ситуаціях метод get() поверне помилку.

Лістинг 13. Приклад типової функції вибірки списка об’єктів по вторинному ключу

```
def get_use_case_list(project_id):
    return UseCase.objects.filter(
        project_link_id=project_id).order_by('name')
```

Функції даного типу використовуються для пошуку списку об’єктів за зовнішнім ключем та являються зручною обгорткою над синтаксисом Django-models. При відсутності об’єктів з заданим критерієм повертають порожній список.

Лістинг 14. Приклад типової функції створення об’єкта

```
def add_actor(project_id, name):
    Actor.objects.create(name=name, project_link_id=project_id)
```

Функції даного типу використовуються для створення об’єктів в системі. Можуть приймати різну кількість аргументів, залежно від кількості полів в сутності та їх задіяності при створенні, якщо наявні необов’язкові поля.

Лістинг 15. Приклад типової функції редагування об’єкта

```
def edit_actor(actor_id, actor_name):
    actor = get_actor_by_id(actor_id)
    actor.name = actor_name
    actor.save()
```

Такі функції мають стандартну послідовність дій: ініціювати знаходження об’єкта по ідентифікатору, присвоїти потрібні значення полям, зберегти зміни. Оскільки пошук об’єкта виконується за допомогою методу get() на використання таких функцій накладається певне обмеження, а саме

те, що об’єкт за заданим ідентифікатором гарантовано має існувати та бути єдиним.

Лістинг 16. Приклад типової функції видалення об’єкта

```
def delete_actor(actor_id):
    actor = get_actor_by_id(actor_id)
    actor.delete()
```

Даний тип функцій завжди приймає один аргумент, а саме ідентифікатор об’єкта. Мають ті ж обмеження на використання, що й функції редагування.

Лістинг 17. Реалізація створення варіанту використання

```
def add_use_case(name, purpose, actor_id, project_id):
    use_case = UseCase.objects.create(name=name,
project_link_id=project_id, actor_link_id=actor_id,
actual_level=1, recommended_level=1, purpose=purpose)
    attribute_list = get_attribute_list(project_id)
    for attribute in attribute_list:
        AttributeValue.objects.create(value=1,
use_case_link_id=use_case.id, attribute_link_id=attribute.id)
```

У даному випадку створення варіанту використання включає в себе заповнення його списку значень атрибутів. Даний варіант реалізації дозволяє спростити функції відображення та обчислення рекомендованого рівня деталізації. Крім того при створенні варіанту використання він записується в реєстр ВВ та ще не має специфічних полів, що застосовуються в шаблонах специфікацій. Тобто природньо що при створенні варіанта використання він автоматично матиме перший рівень деталізації та мінімальні значення атрибутів

Лістинг 18. Реалізація створення атрибуту

```
def add_attribute(project_id, name, weigh, gradation_level):
    attribute = Attribute.objects.create(
        name=name, weigh=weigh, gradation_level=gradation_level,
project_link_id=project_id)
    use_case_list = get_use_case_list(project_id)
    for use_case in use_case_list:
        AttributeValue.objects.create(value=1,
use_case_link_id=use_case.id, attribute_link_id=attribute.id)
```

Для коректної роботи системи, при створенні атрибуту, відповідно до кожного наявного варіанта використання виставляються його мінімальне значення. Це реалізовано в цілях спрощення перевірок і, як наслідок, збільшення швидкодії системи.

Лістинг 19. Реалізація вибірки списку варіантів використання з значеннями атрибутів

```
def get_use_case_list_with_attribute_values(project_id):
    use_case_list = get_use_case_list(project_id)
    use_case_list_with_attribute_values = []
    for use_case in use_case_list:
        tmp = {}
        values = get_attribute_value_list_uc(use_case.id)
        tmp['use_case'] = use_case
        tmp['attribute_values'] = values
        use_case_list_with_attribute_values.append(tmp)
    return use_case_list_with_attribute_values
```

Дана функція проводить вибірку списку варіантів використання та значеннь атрибутів для кожного ВВ в списку. Така реалізація зумовлена тим, що наявна зведена таблиця з полями варіантів використання та значень атрибутів, причому атрибутів може бути довільна кількість. Реалізована на рівні моделі для спрощення функції представлення, яка проводить рендер відповідного HTML шаблону.

Лістинг 20. Реалізація додавання кроку до альтернативи чи виключної ситуації

```
def add_step_from_alt(number, text, alt_id):
    step = Step.objects.create(number=number, text=text)
    ManyToManyAltStep.objects.create(
        step_link_id=step.id, exception_or_alternative_link_id=alt_id,
is_step_to_alt=False)
```

Стандартна реалізація створення кроку використовується для створення кроків основного потоку, в той же час як дана версія використовується для альтернатив та виключних ситуацій. У випадку коли крок породжується від альтернативи чи виключної ситуації він не має прописаного значення зовнішнього ключа сутності UseCase, але має запис в

таблиці ManyToManyAltStep з відображенням напрямку зв'язку. В даному випадку напрямок від альтернативи до кроку. Дана особливість допомагає при реалізації навігації при довільному рівні вкладеності альтернатив та виключних ситуацій.

Лістинг 21. Реалізація додавання альтернативи чи виключної ситуації

```
def add_exception_or_alternative(code, description, name, step_id):
    alt = ExceptionOrAlternative.objects.create(code=code,
description=description, name=name)
    ManyToManyAltStep.objects.create(step_link_id=step_id,
exception_or_alternative_link_id=alt.id, is_step_to_alt=True)
```

Альтернатива при створенні завжди прив'язується до певного кроку, який є для неї батьківським. Так же як і альтернативна функція створення кроку, створює запис в ManyToManyAltStep, проте зв'язок має інший напрямок. Тобто крок є батьківським, а альтернатива породженою. Це реалізовано для реалізації вкладеності та можливості відображення повного шляху.

Лістинг 22. Особливості видалення кроків, альтернатив та виключних ситуацій

```
def delete_alternative(alternative_id):
    alternative = get_exception_or_alternative_by_id(alternative_id)
    alt_step_list = get_step_list_from_alt(alternative.id)
    for alt_step in alt_step_list:
        delete_step(alt_step.id)
    alternative.delete()


def delete_step(step_id):
    step = get_step_by_id(step_id)
    alt_list = get_alt_list_from_step(step.id)
    for alt in alt_list:
        delete_alternative(alt.id)
    step.delete()
```

Оскільки альтернативи та виключні ситуації прямо не пов'язані та мають модифіковане відношення «багато до багатьох» їх видалення проходить за допомогою двох функцій, які викликають одна одну. Основна ідея полягає в тому, що перед тим як видалити крок, потрібно пройти по

всім можливим породженим крокам та видалити їх, в свою чергу для того щоб видалити крок, потрібно спочатку видалити всі вкладені альтернативи. Видалення ж точок розширення, які прив'язані до кроку пройде автоматично, оскільки там уже діє каскадне видалення. За допомогою даної реалізації можливе видалення кроків та альтернатив з довільним рівнем вкладеності.

Лістинг 23. Приклад збереження формсета

```
def save_attribute_value_formset(formset, use_case_id):
    for form in formset:
        attribute_value =
get_attribute_value(form.cleaned_data['attribute_id'], use_case_id)
        attribute_value.value = form.cleaned_data['value']
        attribute_value.save()
```

Даний тип функцій призначений для збереження набору однотипних форм та винесений в функцію моделі для того, щоб спростити відповідні представлення. В цьому випадку зберігається набір значень атрибутів, проте подібна реалізація використовується при збереженні списку кроків.

Лістинг 24. Реалізація встановлення рекомендованого рівня деталізації

```
def set_recommended_level_use_case(use_case_id):
    use_case = get_use_case_by_id(use_case_id)
    attribute_list = get_attribute_list(use_case.project_link_id)
    numerator = 0
    denominator = 0
    for attribute in attribute_list:
        attribute_value = get_attribute_value(attribute.id, use_case.id)
        numerator += (attribute.weigh * attribute_value.value * 4 /
attribute.gradation_level)
        denominator += attribute.weigh
    recommended_level = round(numerator / denominator)
    use_case.recommended_level = recommended_level
    use_case.actual_level = recommended_level
    use_case.save()
```

Саме в даній функції реалізовано спосіб вибору рекомендованого рівня деталізації, що був сформульований в попередній главі. Обчислений рекомендований рівень деталізації записується в відповідне поле сутності варіанту використання та в актуальний рівень деталізації. Рівні деталізації

було розділено на рекомендований, що обчислюється на основі атрибутів та актуальний, який може бути встановлений вручну. Це реалізовано для можливих ситуацій, коли заранння відомо який шаблон специфікації має бути застосованим для обраного варіанту використання.

Лістинг 25. Реалізація вибірки списку унікальних альтернатив та виключних ситуацій для списку кроків

```
def get_unique_alternative_list(step_id):
    step = get_step_by_id(step_id)
    if step.use_case_link_id is None:
        alternative = get_parent_alt_from_step(step.id)
        step_list = get_step_list_from_alt(alternative.id)
    else:
        use_case = get_use_case_by_id(step.use_case_link_id)
        step_list = get_step_list_from_use_case(use_case.id)
    step_list_a = step_list_with_alts(step_list)
    alt_ids = []
    for step in step_list_a:
        alt_list = step['alt_list']
        for alt in alt_list:
            if alt.id not in alt_ids:
                alt_ids.append(alt.id)
    alt_list = ExceptionOrAlternative.objects.filter(id__in=alt_ids)
    return alt_list
```

Інколи трапляються ситуації, коли альтернатива чи виключна ситуація може спрацювати не на одному кроці, а на кількох. Для уникнення дублювання було реалізовано механіку прив’язки існуючої альтернативи чи виключної ситуації до обраного кроку. Дана функція надає набір унікальних об’єктів які потім використовуються при прив’язці однієї з них до певного кроку.

Лістинг 26. Реалізація прив’язки та відв’язки альтернатив та виключних ситуацій

```
def link_alternative(step_id, alternative_id):
    step = get_step_by_id(step_id)
    alternative = get_exception_or_alternative_by_id(alternative_id)
    ManyToManyAltStep.objects.create(
        step_link_id=step.id,
exception_or_alternative_link_id=alternative.id, is_step_to_alt=True)


def unlink_alternative(step_id, alternative_id):
    step = get_step_by_id(step_id)
    alternative = get_exception_or_alternative_by_id(alternative_id)
```

```
    records = ManyToManyAltStep.objects.filter(
        step_link_id=step.id,
exception_or_alternative_link_id=alternative.id, is_step_to_alt=True)
    for record in records:
        record.delete()
```

Дані функції реалізують прив’язку та відв’язку існуючих альтернатив та виключних ситуацій до певного кроку. Вони отримують ідентифікатори кроку та альтернативи від відповідних функцій представлення, які в свою чергу мають, обрану користувачем із списку, альтернативу чи виключну ситуацію.

Лістинг 27. Функції які використовуються для побудови частини навігаційної панелі

```
def get_parent_step_from_alt(alt_id):
    record =
ManyToManyAltStep.objects.filter(exception_or_alternative_link_id=alt_id,
is_step_to_alt=True)[:1]
    return get_step_by_id((record[0]).step_link_id)


def get_parent_alt_from_step(step_id):
    record = ManyToManyAltStep.objects.filter(step_link_id=step_id,
is_step_to_alt=False)[:1]
    return
get_exception_or_alternative_by_id((record[0]).exception_or_alternative_lin
k_id)
```

Дані функції знаходять одну з батьківських сутностей для кроку та альтернативи відповідно. Це використовується при відображенні повного шляху від списку проектів, і до альтернативи чи виключної ситуації яка має довільний рівень вкладеності.

3.3.2 Аналіз функціональності та основних типів функцій представлення

Автоматичну адаптивність інтерфейсу користувача, відповідно до типу користувача та обраного рівня деталізації специфікації, було реалізовано на рівні шаблонів. Розглянемо основну функціональність розробленого програмного продукту та розпочнемо з того, що побачить

користувач чи адміністратор після авторизації, а саме список проектів в системі. Приклади відображення списку проектів для адміністратора та для користувача можна побачити на рис. 7 та рис. 8.

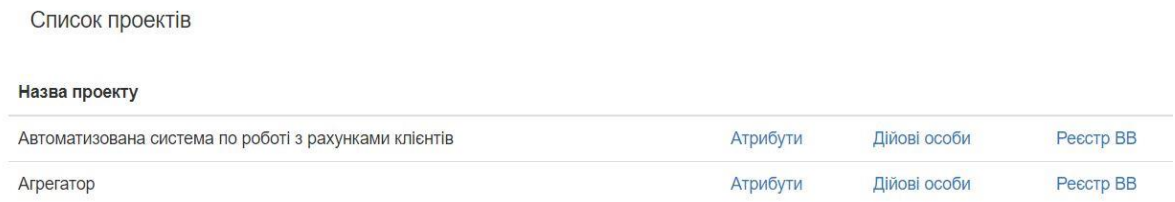


Рис. 7. Список проектів з точки зору адміністратора

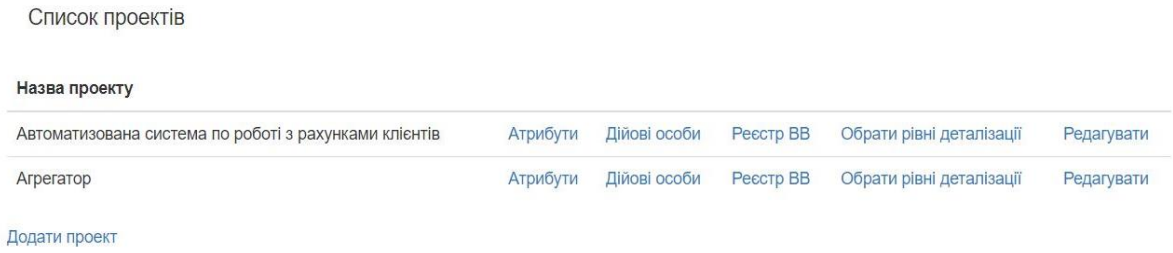


Рис. 8. Список проектів з точки зору користувача

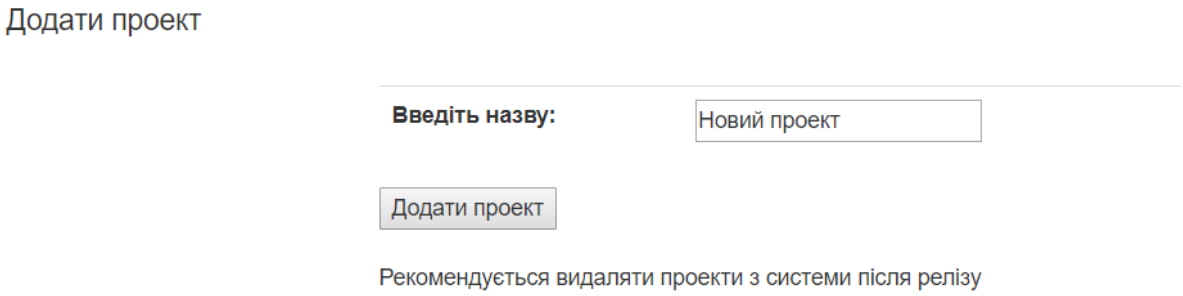


Рис. 9. Форма створення сутності проекту

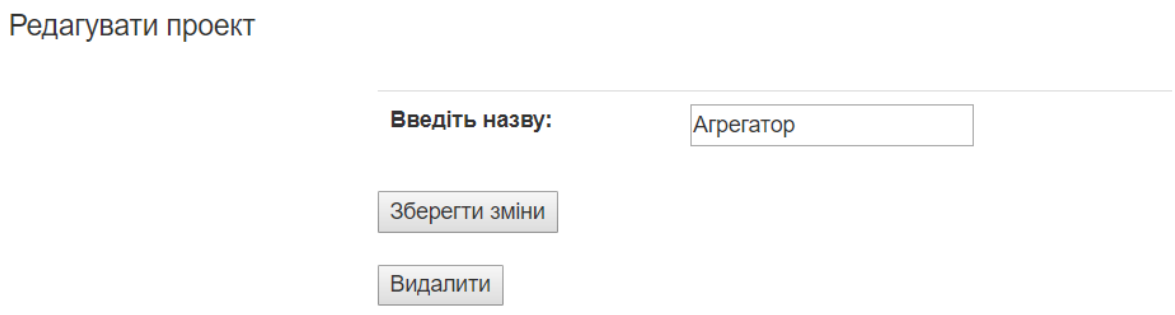


Рис. 10. Форма редагування сутності проекту

З сторінки з відображенням списку проектів користувач може переглянути список атрибутів, дійових осіб та реєстр варіантів використання. Адміністратор додатково може, крім редагування назви проекту, ще перейти на сторінку з встановленням рівнів деталізації. Приклади відображення списку атрибутів для адміністратора та користувача можна побачити на рис. 11 та рис. 12 відповідно. В свою чергу приклади відображення списку дійових осіб можна побачити на рис. 15 та рис. 16 залежно від типу користувача.

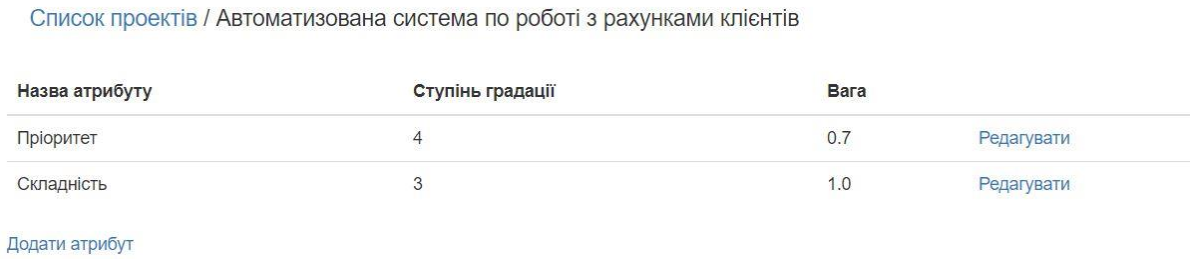


Рис. 11. Список атрибутів з точки зору адміністратора

Рис. 12. Список атрибутів з точки зору користувача

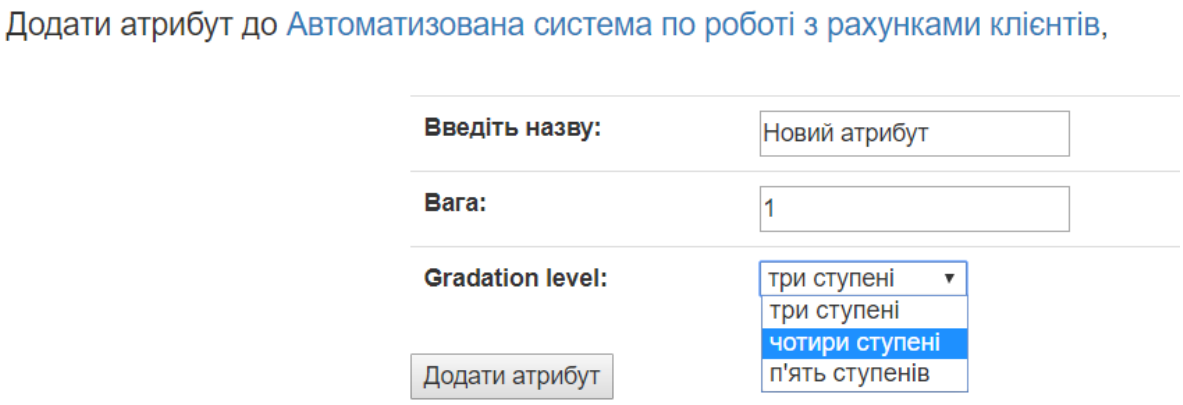


Рис. 13. Форма створення нового атрибута

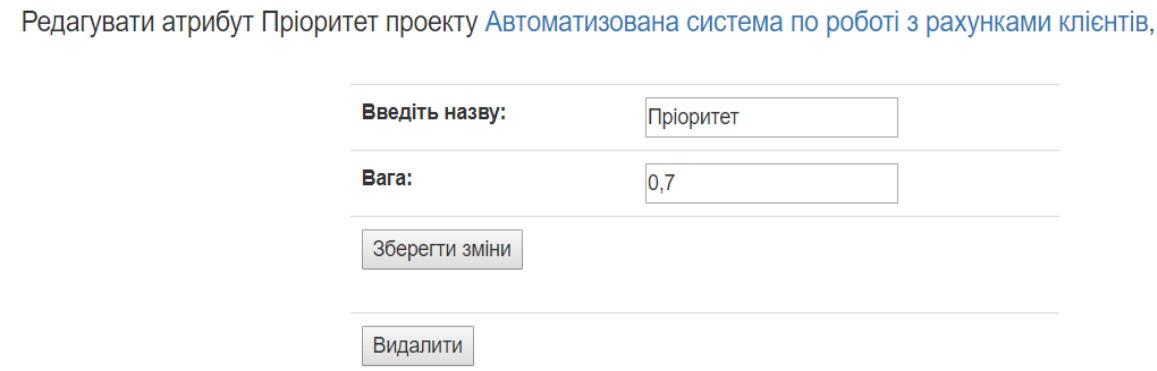


Рис. 14. Форма редагування існуючого атрибута

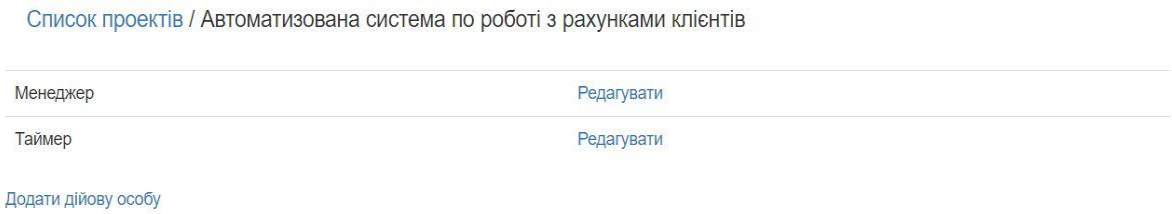


Рис. 15. Список дійових осіб з точки зору адміністратора

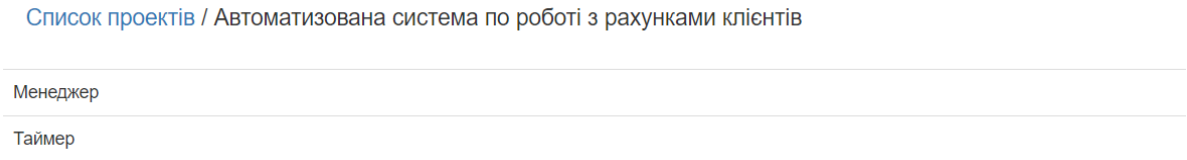


Рис. 16. Список дійових осіб з точки зору користувача

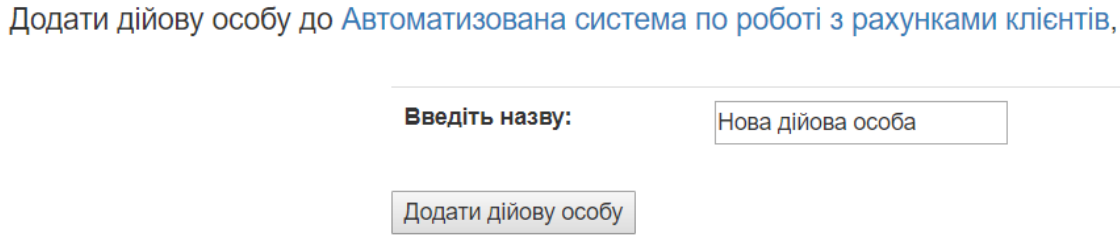


Рис. 17. Форма створення нової дійової особи

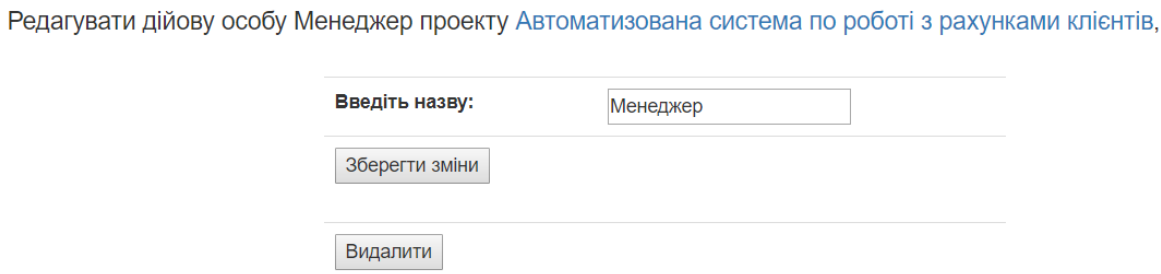


Рис. 18. Форма редагування існуючої дійової особи

Реєстр варіантів використання для кожного ВВ відображає його назву, дійову особу, що є його ініціатором, її ціль та актуальний рівень деталізації. Запис в реєстрі варіантів використання є обов'язковим для кожного ВВ який прийнято до реалізації Тобто усі наявні в системі варіанти використання за замовчуванням мають перший рівень деталізації та в майбутньому, після вибору рівня деталізації, можуть бути деталізованими за певними шаблонами, прийнятими в системі. Приклади відображення реєстру варіантів використання наведено на рис. 19 для адміністратора та на рис. 20 для користувача.

Лістинг 28. Функції представлення реєстру варіантів використання

```
@login_required
def use_case_register(request, project_id):
    project = models.get_project_by_id(project_id)
    context = {}
    use_case_list_with_actors =
models.get_use_case_list_with_actors(project_id)
    context['use_case_list_with_actors'] = use_case_list_with_actors
    context['project'] = project
    return render(request, 'spec/use_case_register.html', context)
```

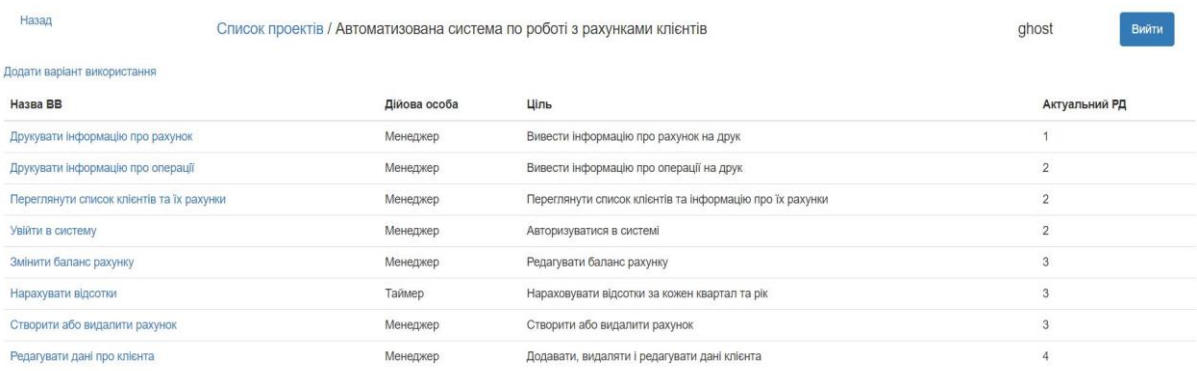
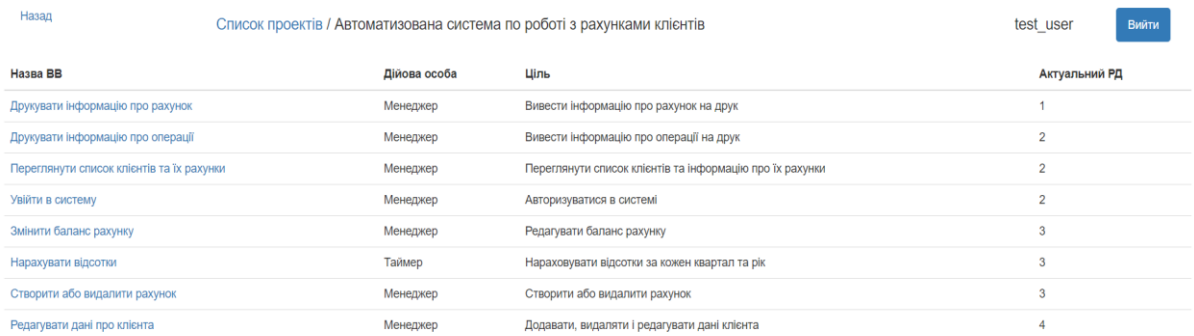


Рис. 19. Реєстр варіантів використання з точки зору адміністратора



Назва ВВ	Дійова особа	Ціль	Актуальний РД
Друкувати інформацію про рахунок	Менеджер	Вивести інформацію про рахунок на друк	1
Друкувати інформацію про операції	Менеджер	Вивести інформацію про операції на друк	2
Переглянути список клієнтів та їх рахунки	Менеджер	Переглянути список клієнтів та інформацію про їх рахунки	2
Увійти в систему	Менеджер	Авторизуватися в системі	2
Змінити баланс рахунку	Менеджер	Редагувати баланс рахунку	3
Нарахувати відсотки	Таймер	Нараховувати відсотки за кожен квартал та рік	3
Створити або видалити рахунок	Менеджер	Створити або видалити рахунок	3
Редагувати дані про клієнта	Менеджер	Додавати, видаляти і редагувати дані клієнта	4

Рис. 20. Реєстр варіантів використання з точки зору користувача

Адміністратору доступне визначення рівня деталізації, причому як рекомендованого рівня а й актуального. Крім того, на даній сторінці можна переглянути описи варіантів використання та додати нові.

Лістинг 29. Функція представлення сторінки керування рівнями деталізації варіантів використання

```
@login_required
def use_case_list(request, project_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    project = models.get_project_by_id(project_id)
    attribute_list = models.get_attribute_list(project.id)
    context = {}
    use_case_list_with_val =
models.get_use_case_list_with_attribute_values(project_id)
    context['use_case_list_with_val'] = use_case_list_with_val
    context['project'] = project
    context['attribute_list'] = attribute_list
    return render(request, 'spec/use_case_list.html', context)
```

Назва ВВ	Актуальний РД	Рекомендований РД	Пріоритет	Складність
Друкувати інформацію про операції	2	2	1	2
Друкувати інформацію про рахунок	1	1	1	1
Змінити баланс рахунку	3	3	2	3
Нарахувати відсотки	3	3	1	3
Переглянути список клієнтів та їх рахунки	2	2	4	1
Редагувати дані про клієнта	4	4	4	3
Створити або видалити рахунок	3	3	4	2
Увійти в систему	2	2	3	1

Рис. 21. Сторінка керування рівнями деталізації

Лістинг 30. Функція представлення сторінки вибору рівня деталізації на основі значень атрибутів

```
@login_required
def choose_level_of_detalisation(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    attributeFormSet = formset_factory(form=EditAttributeValueForm,
extra=0)
    attribute_list = models.get_attribute_list(use_case.project_link_id)
    initial_data = [{'attribute_name': q.name, 'attribute_id': q.id} for q
in attribute_list]
    formset = attributeFormSet(initial=initial_data)
    for form, attribute in zip(formset, attribute_list):
        if 5 == attribute.gradation_level:
            form.fields['value'].choices = CHOICES_5
        elif 4 == attribute.gradation_level:
            form.fields['value'].choices = CHOICES_4
        else:
            form.fields['value'].choices = CHOICES_3
    if request.method == 'POST':
        formset = attributeFormSet(request.POST)
        for form, attribute in zip(formset, attribute_list):
            if 5 == attribute.gradation_level:
                form.fields['value'].choices = CHOICES_5
            elif 4 == attribute.gradation_level:
                form.fields['value'].choices = CHOICES_4
            else:
                form.fields['value'].choices = CHOICES_3
        if formset.is_valid():
            models.save_attribute_value_formset(formset, use_case.id)
            models.set_recommended_level_use_case(use_case.id)
        return redirect('/spec/use_case_list/%i/' %
use_case.project_link_id)
    context = {'formset': formset, 'use_case': use_case, }
    return render(request, 'spec/choose_level_of_detalisation.html',
context)
```

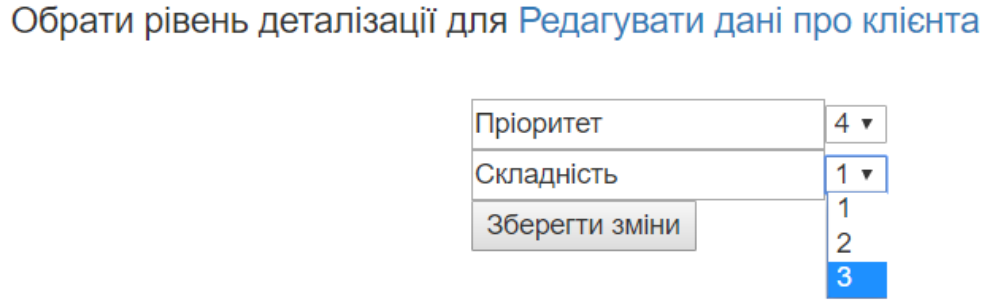


Рис. 22. Сторінка встановлення значень атрибутів для вибору РД

Лістинг 31. Функція представлення сторінки ручного вибору РД

```
@login_required
def set_level_of_detalisation(request, use_case_id):
    user = auth.get_user(request)
```

```
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    if request.method == 'POST':
        form = EditLevelOfDetalisation(request.POST)
        if form.is_valid():
            models.set_actual_level_use_case(use_case.id,
form.cleaned_data['level_of_detalisation'])
            return redirect('/spec/use_case_list/%i/' %
use_case.project_link_id)
    else:
        form = EditLevelOfDetalisation()
        context = {'form': form, 'use_case': use_case}
        return render(request, 'spec/set_level_of_detalisation.html',
context)
```

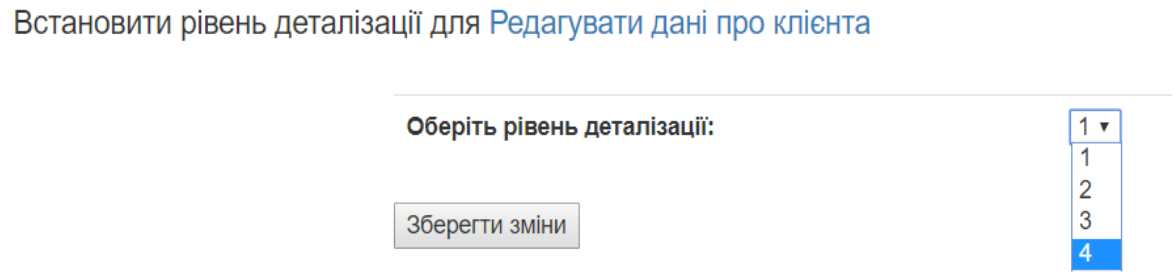


Рис. 23. Сторінка ручного встановлення рівня деталізації

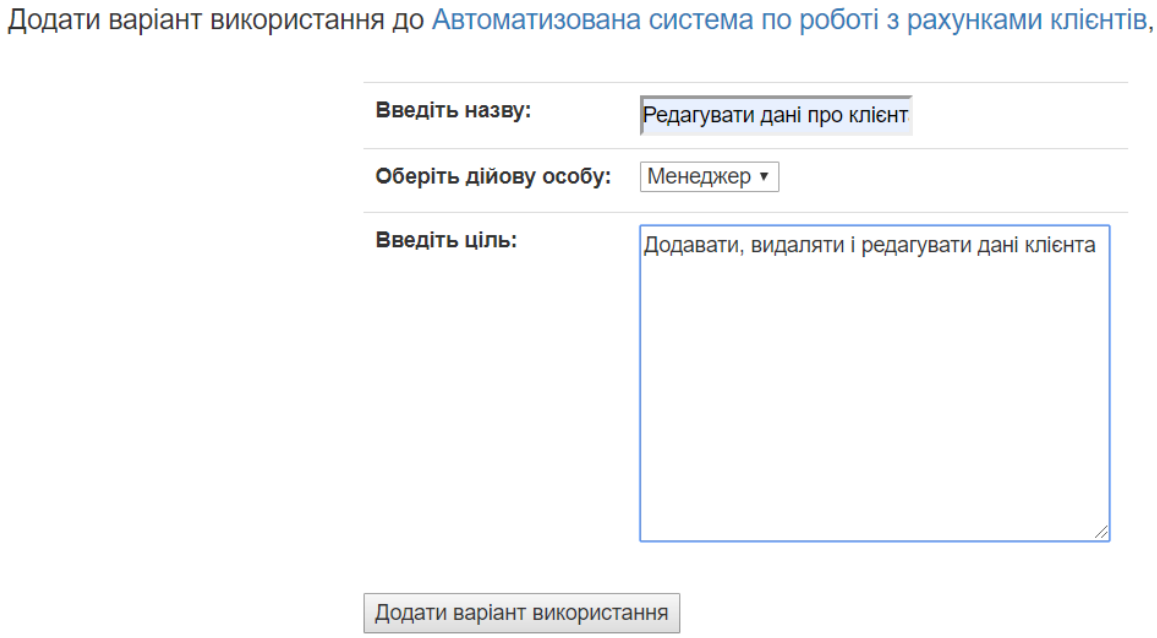


Рис. 24. Форма створення нової сутності варіанта використання

Адаптивність відображення специфікацій реалізовано на рівні шаблонів та залежить від типу користувача та обраного рівня деталізації. Це

суттєво підвищує зручність та простоту використання, оскільки користувачу автоматично надається потрібний шаблон для написання специфікації, що включає в себе створення модулів, які відповідають обраному шаблону. Заповнення усіх поданих полів є обов'язковим, звичайно якщо такі елементи передбачені самим варіантом використання, який описується.

Лістинг 32. Функція представлення сторінки специфікації

```
@login_required
def use_case_description(request, use_case_id):
    use_case = models.get_use_case_by_id(use_case_id)
    trigger_list = models.get_trigger_list(use_case.id)
    context = {}
    context['use_case'] = use_case
    context['trigger_list'] = trigger_list
    context['actor'] = models.get_actor_by_id_safe(use_case.actor_link_id)
    context['project'] = models.get_project_by_id(use_case.project_link_id)
    step_list = models.get_step_list_from_use_case(use_case.id)
    step_list_with_alt = models.step_list_with_alts(step_list)
    context['step_list'] = step_list_with_alt
    conditions = models.get_condition_list(use_case.id)
    pre = []
    post = []
    spec = []
    for condition in conditions:
        if 1 == condition.type:
            pre.append(condition)
        elif 2 == condition.type:
            post.append(condition)
        else:
            spec.append(condition)
    context['pre_conditions'] = pre
    context['post_conditions'] = post
    context['spec_conditions'] = spec
    return render(request, 'spec/use_case_description.html', context)
```

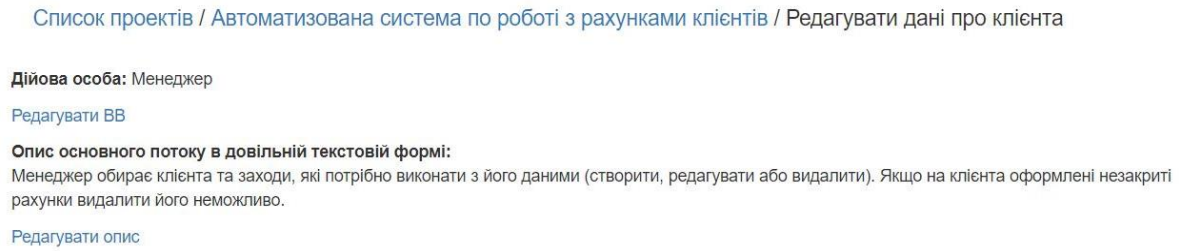


Рис. 25. Приклад специфікації другого рівня деталізації з точки зору адміністратора

Список проектів / Автоматизована система по роботі з рахунками клієнтів / Редагувати дані про клієнта

Дійова особа: Менеджер

Опис основного потоку в довільній текстовій формі:
Менеджер обирає клієнта та заходи, які потрібно виконати з його даними (створити, редагувати або видалити). Якщо на клієнта оформлені незакриті рахунки видалити його неможливо.

Рис. 26. Приклад специфікації другого рівня деталізації з точки зору користувача

Список проектів / Автоматизована система по роботі з рахунками клієнтів / Редагувати дані про клієнта

Дійова особа: Менеджер

Передумови:

- Перед початком виконання даного варіанту використання менеджер повинен увійти в систему (Редагувати)

Короткий опис: Даний варіант використання дозволяє менеджеру додавати, видаляти і редагувати дані клієнта.

Тригери:

- Даний варіант використання починає виконуватися, коли менеджер вирішує змінити дані про клієнта (Редагувати)

Додати тригер

Редагувати ВВ

Керувати кроками ВВ

№	Крок	Виключні ситуації/ альтернативи
1	Система запитує потрібні дії (створити клієнта, змінити дані про клієнта, видалити клієнта)	
2	Менеджер обирає операцію	
	. . .	
3.2.1	Система виводить поточну інформацію про клієнта	
3.2.2	Менеджер редагує інформацію та підтверджує її збереження	A1 A2
3.2.3	Система зберігає та оновлює дані про клієнта	A2
3.3	Якщо менеджер обрав «видалити клієнта»	
3.3.1	Система виводить поточну інформацію про клієнта та запитує у менеджера підтвердження видалення клієнта.	
3.3.2	Менеджер підтверджує видалення	A3
3.3.3	Система видаляє дані про клієнта	A4

Додати умову

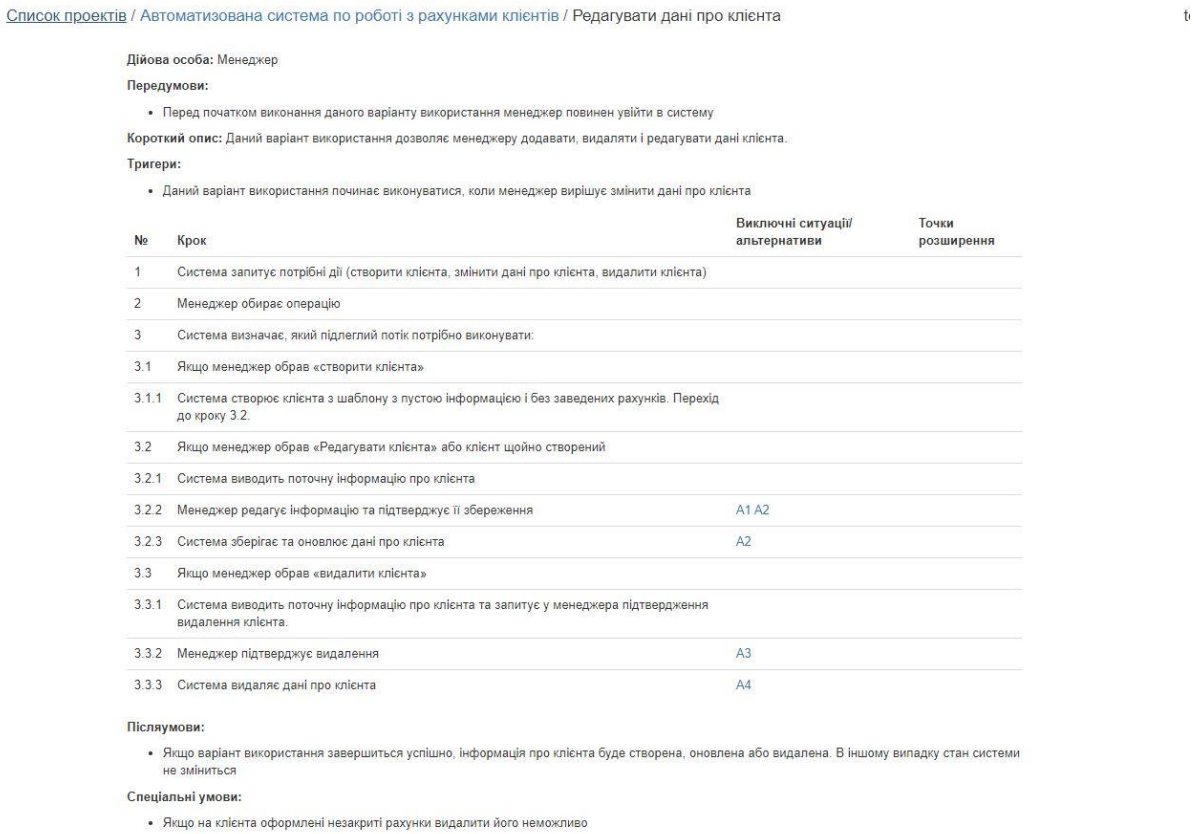
Післяумови:

- Якщо варіант використання завершиться успішно, інформація про клієнта буде створена, оновлена або видалена. В іншому випадку стан системи не зміниться (Редагувати)

Спеціальні умови:

- Якщо на клієнта оформлені незакриті рахунки видалити його неможливо (Редагувати)

Рис. 27. Приклад специфікації третього рівня деталізації з точки зору адміністратора



Список проектів / Автоматизована система по роботі з рахунками клієнтів / Редагувати дані про клієнта

Дійова особа: Менеджер

Передумови:

- Перед початком виконання даного варіанту використання менеджер повинен увійти в систему

Короткий опис: Даний варіант використання дозволяє менеджеру додавати, видаляти і редагувати дані клієнта.

Тригери:

- Даний варіант використання починає виконуватися, коли менеджер вирішує змінити дані про клієнта

№	Крок	Виключні ситуації/ альтернативи	Точки розширення
1	Система запитує потрібні дії (створити клієнта, змінити дані про клієнта, видалити клієнта)		
2	Менеджер обирає операцію		
3	Система визначає, який підлеглий потік потрібно виконувати:		
3.1	Якщо менеджер обрав «створити клієнта»		
3.1.1	Система створює клієнта з шаблону з пустою інформацією і без заведених рахунків. Перехід до кроку 3.2.		
3.2	Якщо менеджер обрав «Редагувати клієнта» або клієнт щойно створений		
3.2.1	Система виводить поточну інформацію про клієнта		
3.2.2	Менеджер редагує інформацію та підтверджує її збереження	A1 A2	
3.2.3	Система зберігає та оновлює дані про клієнта	A2	
3.3	Якщо менеджер обрав «видалити клієнта»		
3.3.1	Система виводить поточну інформацію про клієнта та запитує у менеджера підтвердження видалення клієнта.		
3.3.2	Менеджер підтверджує видалення	A3	
3.3.3	Система видаляє дані про клієнта	A4	

Післяумови:

- Якщо варіант використання завершиться успішно, інформація про клієнта буде створена, оновлена або видалена. В іншому випадку стан системи не зміниться

Спеціальні умови:

- Якщо на клієнта оформлені незакриті рахунки видалити його неможливо

Рис. 27. Приклад специфікації четвертого рівня деталізації з точки зору користувача

Представленння альтернативи відрізняється від представлення варіанта використання складною навігацією, шлях якої обраховується при запиті до сторінки. Це потрібно для реалізації можливості вкладеності альтернатив та виключних ситуацій, оскільки в класичному вигляді специфікацій також допускається певна вкладеність. Проте в монолітному представленні специфікації використання вкладених альтернатив та виключних ситуацій вважається поганою практикою, оскільки це занадто сильно ускладнює роботу з специфікацією. Цю проблему там вирішують за допомогою особливої нумерації, яка також передбачена у розробленому програмному продукті, чи винесенням вкладеності в точки розширення. Проте при модульному поданні це не є проблемою, оскільки інфомація подається блоками, а відображення повного шляху надає чітке представлення про позиціонування відображуваного модуля.

Лістинг 33. Функція представлення сторінки альтернативи

```
@login_required
def alt_description(request, alt_id):
    alt = models.get_exception_or_alternative_by_id(alt_id)
    context = {}
    context['alternative'] = alt
    step_list = models.get_step_list_from_alt(alt.id)
    step_list_with_alt = models.step_list_with_alts(step_list)
    context['step_list'] = step_list_with_alt
    alternative_list = []
    parent_step = models.get_parent_step_from_alt(alt.id)
    is_use_case_back = 1
    use_case_id = parent_step.use_case_link_id
    while parent_step.use_case_link_id is None:
        is_use_case_back = 0
        tmp = models.get_parent_alt_from_step(parent_step.id)
        alternative_list.append(tmp)
        parent_step = models.get_parent_step_from_alt(tmp.id)
        use_case_id = parent_step.use_case_link_id
    context['is_use_case_back'] = is_use_case_back
    use_case = models.get_use_case_by_id(use_case_id)
    context['use_case'] = use_case
    if is_use_case_back == 1:
        context['back_id'] = use_case.id
    else:
        back = alternative_list[0]
        context['back_id'] = back.id
    alternative_list.reverse()
    context['nav_list'] = alternative_list
    context['project'] = models.get_project_by_id(use_case.project_link_id)
    return render(request, 'spec/alt_description.html', context)
```

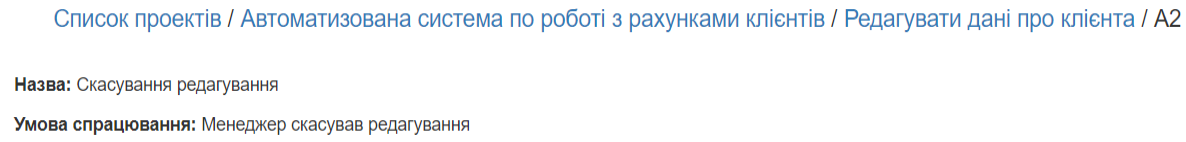


Рис. 28. Приклад альтернативи третього рівня деталізації з точки зору користувача

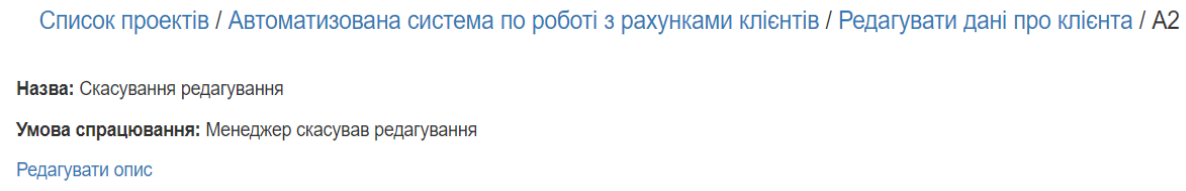


Рис. 29. Приклад альтернативи третього рівня деталізації з точки зору адміністратора

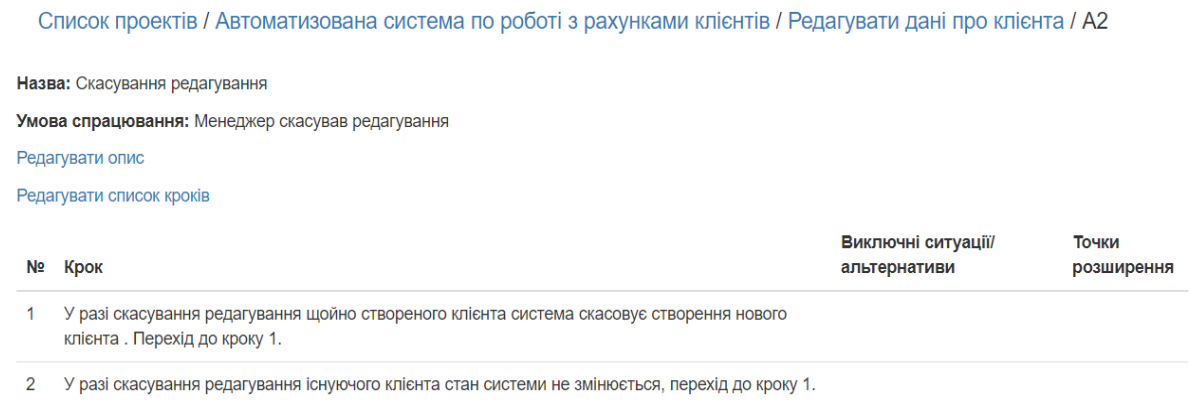


Рис. 30. Приклад альтернативи четвертого рівня деталізації з точки зору адміністратора

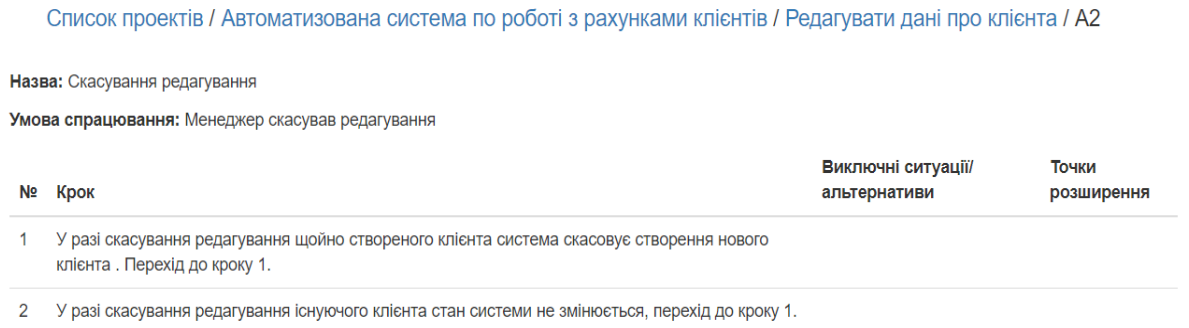


Рис. 31. Приклад альтернативи четвертого рівня деталізації з точки зору користувача

Лістинг 34. Функція представлення сторінки редагування сутності варіанта використання

```
@login_required
def edit_use_case(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    project = models.get_project_by_id(use_case.project_link_id)
    if request.method == 'POST':
        form = EditUseCaseForm(request.POST)
        form.fields['actor'].choices = [(actor.id, actor.name) for actor in
Actor.objects.filter(
            project_link_id=project.id)]
        if form.is_valid():
            models.edit_use_case(
                use_case_id, form.cleaned_data['name'],
form.cleaned_data['description'], form.cleaned_data['purpose'],
                form.cleaned_data['actor'])
            return redirect('/spec/use_case_description/%i/' % use_case.id)
    else:
        form = EditUseCaseForm(
```

```
            initial={'name': use_case.name, 'description':
use_case.description, 'purpose': use_case.purpose})
        form.fields['actor'].choices = [(actor.id, actor.name) for actor in
Actor.objects.filter(
            project_link_id=project.id)]
        context = {'form': form, 'use_case': use_case, 'project': project}
        return render(request, 'spec/edit_use_case.html', context)
```

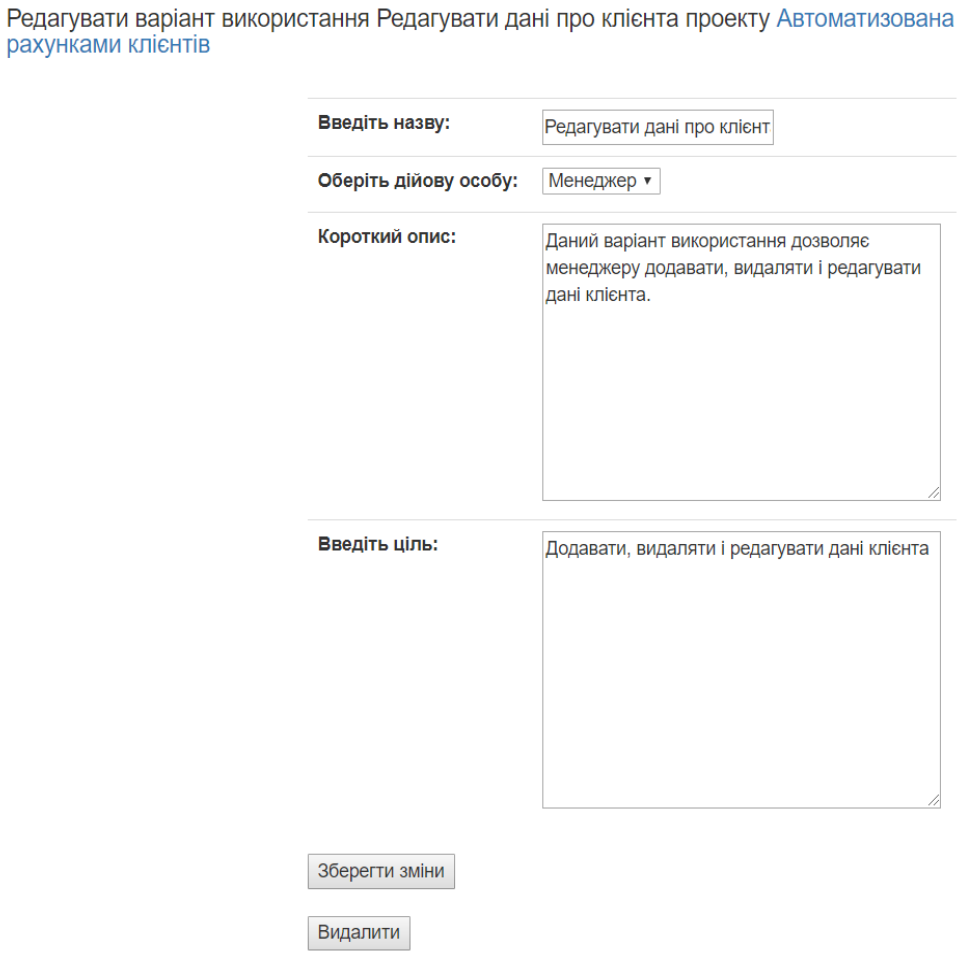


Рис. 32. Форма редагування сутності варіанта використання

Редагування списку кроків реалізовано на основі Django-formsets, що в свою чергу являє набір однотипних форм, збереження яких проводиться за один запит до сервера. Проте дана технологія не задовільняла в повній мірі поставлену задачу, оскільки формсет не передбачає динамічного додавання форм та їх видалення. Максимум можливо задати поле extra, що дозволить на кожному запускові сторінки додавати до наявного набору форм, фіксовану кількість пустих. Проте такий підхід взагалі не підходить під дану задачу та суттєво ускладнює валідацію форм, тому на рівні

шаблону було підключено власні функції для додавання та видалення форм. Також при виводі списку форм додано колонки з альтернативами та точками розширення, проте вони відображаються без границь таблиці, через обмеження відображення формсета. Проте це суттєво не зменшує ергономіку використання, тому допустимо для даної задачі. Проте для подальшого розвитку потрібно розробити чи адаптувати іншу технологію, яка краще підійде в плані ергономіки та зменшить навантаження на сервер, оскільки додавання кроків хоч і виглядає миттєвим та здається враження що крок додано прямо на сторінці, проте насправді це проходить через виклик функції, яка створює пустий крок та перенаправляє запит назад на сторінку. Тобто в результаті додавання нового кроку сторінка перезавантажується. Тому також рекомендовано перед додаванням кроку зберегти поточний стан списку кроків відповідною кнопкою.

Лістинг 35. Функції представлення сторінок редагування списку кроків для варіантів використання та альтернатив

```
@login_required
def edit_step_list_use_case(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    project = models.get_project_by_id(use_case.project_link_id)
    StepFormSet = formset_factory(form=EditStepForm, extra=0)
    qset = models.get_step_list_from_use_case(use_case.id)
    step_list_with_alt = models.step_list_with_alts(qset)
    initial_data = [{'text': q.text, 'number': q.number} for q in qset]
    id_list = [{'id': q.id} for q in qset]
    formset = StepFormSet(initial=initial_data)
    if request.method == 'POST':
        formset = StepFormSet(request.POST, initial=initial_data)
        if formset.is_valid():
            models.save_step_formset_with_ids(formset, id_list)
        return redirect('/spec/edit_step_list_use_case/%i/' % use_case.id)
    context = {'formset': formset, 'use_case': use_case, 'step_list':
step_list_with_alt, 'project': project, }
    return render(request, 'spec/edit_step_list_use_case.html', context)


@login_required
def edit_step_list_alt(request, alt_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    alternative = models.get_exception_or_alternative_by_id(alt_id)
    alternative_list = []
```

```
    parent_step = models.get_parent_step_from_alt(alternative.id)
    use_case_id = parent_step.use_case_link_id
    while parent_step.use_case_link_id is None:
        tmp = models.get_parent_alt_from_step(parent_step.id)
        alternative_list.append(tmp)
        parent_step = models.get_parent_step_from_alt(tmp.id)
        use_case_id = parent_step.use_case_link_id
    alternative_list.reverse()
    use_case = models.get_use_case_by_id(use_case_id)
    StepFormSet = formset_factory(form=EditStepForm, extra=0)
    qset = models.get_step_list_from_alt(alternative.id)
    step_list_with_alt = models.step_list_with_alts(qset)
    initial_data = [{'text': q.text, 'number': q.number} for q in qset]
    id_list = [{'id': q.id} for q in qset]
    formset = StepFormSet(initial=initial_data)
    if request.method == 'POST':
        formset = StepFormSet(request.POST, initial=initial_data)
        if formset.is_valid():
            models.save_step_formset_with_ids(formset, id_list)
        return redirect('/spec/edit_step_list_alt/%i/' % alternative.id)
    context = {'formset': formset,
               'alternative': alternative,
               'nav_list': alternative_list,
               'step_list': step_list_with_alt,
               'use_case': use_case,
               'project':
models.get_project_by_id(use_case.project_link_id),
               }
    return render(request, 'spec/edit_step_list_alt.html', context)
```

Рис. 33. Модифікований формсет редагування списку кроків ВВ

3.4. Висновки до розділу 3

В даному розділі було описано основні інструменти розроблення, детально проаналізовано архітектуру, особливості реалізації та функціональність програмного продукту, що автоматизує створення специфікацій варіантів використання на основі їх атрибутів. Також продемонстровано перехід варіанта використання від реєстру використання та нарощування його деталізації до четвертого рівня деталізації, який є найбільш повним і точним. Цьому сприяє вдало підібраний набір шаблонів та модульність представлення специфікацій. При цьому специфікація розглядається не як монолітна сутність, а як набір певних компонентів, які в свою чергу можна легко додавати, прибирати та змінювати. Крім того, продемонстровано адаптивність інтерфейсу відповідно до типу користувача та обраного рівня деталізації специфікації. Вибір рівня деталізації можливий як з допомогою способу, описаного у другому розділі, так і вручну.

4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Тестування розробленого програмного забезпечення

Тестування програмного продукту проводилося з початку розроблення і закінчилось після фінального етапу тестуванням методом «сірого ящика».

На початковому етапі розроблення тестувалася структура бази даних на можливість реалізації запланованої функціональності. в процесі якого було обрано таку, яка найкраще підходить для розв’язання поставленої задачі.

В процесі написання основних функцій моделі застосовувалися статичне тестування та рефакторинг готового коду. При розробленні представлень та шаблонів HTML сторінок, крім статичного тестування коду використовувалося тестування сірого ящика та тестування всіх компонентів, на які впливає тестована функціональність. Тобто в процесі розроблення було неформально, тобто без текстового оформлення, виконано велику кількість тестових сценаріїв розроблюваної функціональності, щоб практично повністю покрити всі випадки її роботи.

Також велику увагу було приділено тестуванню користувацьких інтерфейсів. Особливу увагу приділено тестуванню навігації, оскільки деревовидний інтерфейс є достатньо складним для користувача, тобто він має завжди бачити позицію, в якій він знаходиться в даний час. Також це накладає певні обмеження на апаратну частину сервера, які описані в наступному розділі.

4.2. Рекомендації до апаратного забезпечення серверної частини

Розроблений програмний продукт працюватиме швидко та без перебоїв, якщо в розробленій системі не буде знаходиться багато паралельних проектів, чи проектів, які вже були завершені. Таким чином для адміністратора системи рекомендується видаляти дані про проекти, реліз яких уже пройшов успішно.

В даному випадку обмеження накладаються не на об’єм даних, а на швидкість роботи з ними, оскільки створювані системою специфікації складаються з малих модулів, вибірку яких потрібно проводити швидко. Для даної задачі чудово підійде невеликий сервер з швидкою зовнішньою пам’яттю. Специфіка роботи з даними передбачає знижені вимоги до швидкодії при записі, проте підвищені при читанні. З цією задачею можуть впоратися швидкі HDD та SSD, причому останні будуть працювати достатньо довговічно, оскільки більшість запитів проходять саме на читання, а не на запис. Це зумовлено не тільки специфікою роботи системи, а й тим, що відображення навігаційних елементів достатньо сильно навантажує сервер запитами, проте без них обійтися практично неможливо, оскільки даний програмний продукт має деревовидний інтерфейс, який є складним для користувача, проте ідеально підходить для розв’язання поставленої задачі.

4.3. Вдосконалення розробленого програмного забезпечення

Вдосконалення розробленого забезпечення доцільно проводити в таких напрямках, як логування змін, повідомлення підписаних користувачів про внесені зміни, можливість повернення до попереднього варіанту специфікацій (елементи системи контролю версій), генерація класичного представлення специфікацій, можливість адміністратору додавати власні поля із вказанням рівня специфікації, на якому воно використовується, введення нових ролей користувачів із різними ступенями доступу.

Логування змін та оповіщення при їх внесенні дозволить оперативно реагувати на зміни в специфікаціях, а те, що реалізовано їх модульне представлення, значно спростить дану задачу. Тобто користувач, що є підписаним на зміни в обраних варіантах використання, отримуватиме відповідні оповіщення на електронну пошту.

Генерація класичного представлення специфікацій корисна в тих випадках, коли специфікація використовується не лише як чітко

структуроване та формалізоване представлення інформації, а й в якості документу.

Крім того можливо реалізувати модифікований метод аналізу ієрархій для атрибутів. Він допоможе визначити вагові коефіцієнти атрибутів та, при їх великій кількості, відкинути найменш суттєві. Сама ж модифікація методу аналізу ієрархій полягає в тому, що порівняння пари атрибутів доцільно проводити в нестрогому вигляді, тобто з наявним станом рівності.

Також деякі інтерфейси потребують доопрацювання в плані ергономіки використання.

4.4. Висновки до розділу 4

У даному розділі було описано методику тестування та на яких етапах які типи тестування застосовувалися. Тобто на початку тестувалася структура бази даних на можливість повної реалізації розроблюваного програмного забезпечення. В процесі розроблення застосовувалося статичне тестування коду та рефакторинг, поки не було користувацького інтерфейсу, потім додатково проводилось тестування методом сірого ящика. У результаті проведених фінального етапу тестування не знайдено помилок в роботі програми.

Також були сформовані обмеження та рекомендації відносно серверної частини. Вимоги до клієнтської частини є достатньо низькими. Крім цього було описано основні шляхи подальшого вдосконалення програмного продукту, що в майбутньому дозволить підвищити зручність використання та розширити сферу його застосування.

5. ПОБУДОВА БІЗНЕС-МОДЕЛІ

5.1. Опис проблеми

Специфікації варіантів використання (ВВ) відіграють велику роль при розробленні програмного забезпечення (ПЗ). Системні аналітики використовують їх для виявлення функціональних вимог, програмісти - для розробки коду, тестувальники – для побудови тестових сценаріїв, технічні письменники – для написання керівництв користувачів. Тому постає питання про організацію ефективної роботи по специфікації ВВ, адже кількість ВВ навіть в проектах середнього масштабу є досить великою.

Процес специфікації варіантів використання є важливим етапом розроблення програмного забезпечення, оскільки він дозволяє отримати чітке уявлення про майбутній програмний продукт. На етапі збору вимог часто трапляються їх неточність та неоднозначність, що ускладнює процес розроблення та вносить додаткові дефекти. Також підвищується ймовірність того, що програмний продукт не буде задовільняти очікуванням зацікавлених осіб. Ці проблеми і вирішуються в процесі специфікації вимог, проте цей етап часто пропускається за браком часу або ресурсів та через велику кількість паперової роботи та ручних обрахунків.

Для невеликих, добре злагоджених командах та відносно простих завданнь, процес специфікації не є обов’язковим, оскільки в таких умовах мінімізується неточність та неоднозначність і без цього процесу. Проте його

Все вище описане можна узагальнити у схемі «Дерева проблем», яка зображена на рис. 34.

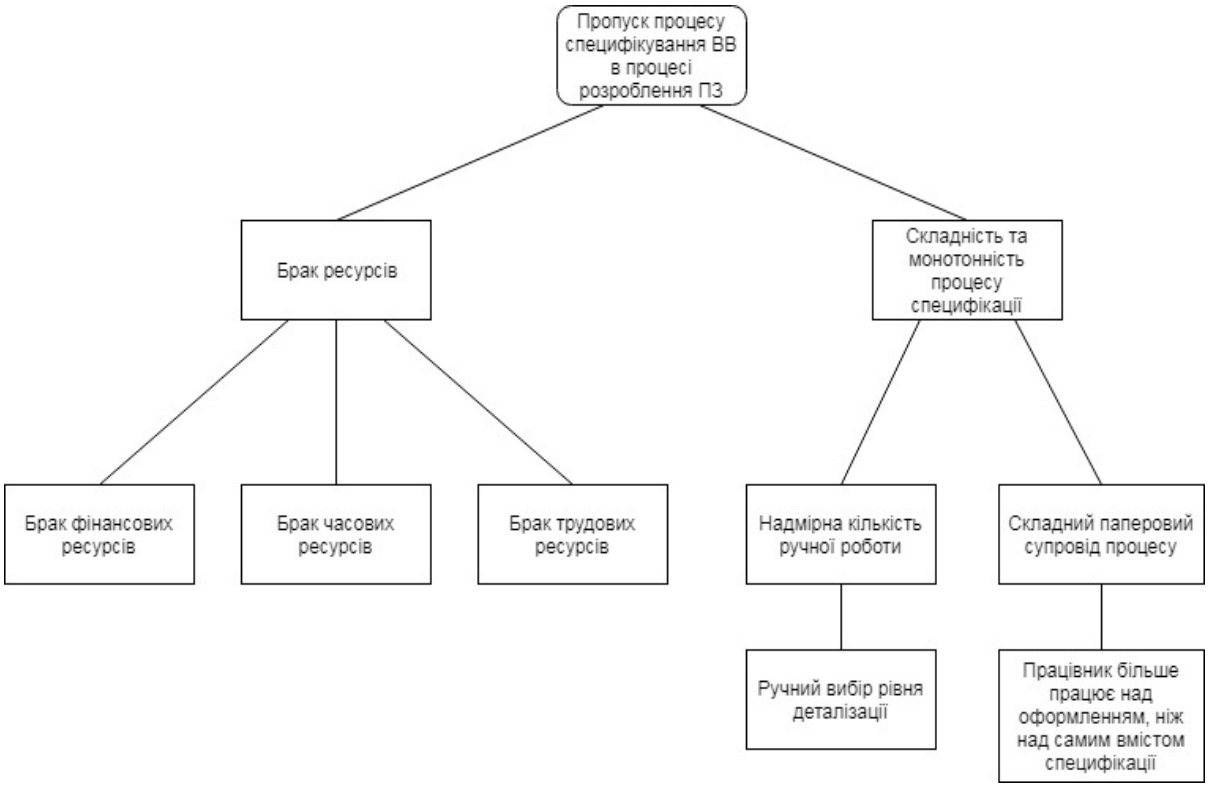


Рис. 34. Дерево проблем

5.2. Зацікавлені сторони

У вирішенні описаної вище проблеми існує кілька зацікавлених сторін.

Найбільш очевидною та впливовою зацікавленою стороною, є розробники програмного забезпечення. Неточність та неоднозначність вимог вносить суттєві дефекти на архітектурному рівні, які складно і дорого усуваються, причому це не завжди можливо чи доцільно. Проте класичний процес специфікації, який вирішує ці проблеми, не завжди задовільняє їх потреби.

Також зацікавленими сторонами є замовники програмного забезпечення у розробників, проте вони не мають суттєвого впливу і здебільшого не уявляють внутрішні процеси при розробленні ПЗ.

У табл. 12 зведено усі групи зацікавлених сторін, їх інтереси, та вплив (міра зацікавленості у вирішенні наявних проблем).

Таблиця 12

Зацікавлені сторони

Зацікавлена сторона	**Інтерес зацікавленої особи**	**Вплив зацікавленої особи**	**Стратегії приваблення зацікавлених сторін**
Розробники програмного забезпечення	Зменшення витрат на усунення архітектурних недоліків та невідповідностей вимогам	Високий	Проведення презентації для представників зацікавлених осіб. Участь у спеціалізованих виставках та форумах
Замовники програмного забезпечення	Отримати програмний продукт який вони очікують від розробників	Низький	

5.3. Комерційне рішення. Основні характеристики

Відповідно до вищезазначених проблем, можна описати кінцевий продукт, що має їх вирішувати. Даний програмний продукт буде реалізовувати автоматизований процес специфікування варіантів використання на основі методу Коберна, описаному в розділі 1. Дана модифікація методу дозволяє значно покращити гнучкість алгоритму вибору рівня деталізації та автоматизувати його. Завдяки автоматичній ініціалізації 1-цями вагових коефіцієнтів відповідних атрибутів, для працівника, якому не потрібні додаткові налаштування не буде витрачати час на них. Це дозволить знизити витрати часу та ресурсів на створення специфікацій варіантів використання та прибрати частину ручної роботи.

5.4. Конкурентні переваги рішення

Описана модифікація методу та відповідне програмне рішення дозволяє одночасно:

- підвищити гнучкість алгоритму вибору рівня деталізації;
- автоматизований вибір рівня деталізації;
- полегшення внесення змін;
- автоматичне оповіщення при внесенні змін.

Крім вищезазначених переваг описаний описаний програмний продукт є веб-орієнтованим.

5.5. Клієнти. Сегменти ринку споживання

Як вже зазначалося, клієнтами для даного програмного рішення є розробники програмного забезпечення. Основна їх задача – розробити програмне забезпечення з мінімальною кількістю дефектів, яке б задовольняло очікування та потреби замовників чи зацікавлених осіб.

На сьогодні йдуть значні витрати ресурсів, як фінансових так і трудових, на подолання дефектів програмного забезпечення, причому найдорожче обходиться ліквідація дефектів на рівні архітектури ПЗ, тому що це займає надзвичайно великий об’єм робіт, на не завжди є можливим чи доцільним, тобто інколи простіше створити новий програмний продукт, ніж виправляти наявний, прицьому всі затрати не окупаються. Це являється наслідком неякісно проведеного процесу специфікації варіантів використання або його повного пропуску.

Потенційними клієнтами є компанії які спеціалізуються на розробленні програмного забезпечення, проте даний продукт не потрібен компаніям які добре обходяться без процесу специфікації, тобто мають невелику, добре організовану команду та займаються задачами невеликого масштабу.

5.6. Унікальна ціннісна пропозиція

Ціннісна пропозиція – це пояснення того, як продукт вирішує проблему. Його можна скласти за формулою:

Ціннісне пропозицію = Проблема + Рішення / Продукт.

У дереві проблем було виділено низку проблем, а у зацікавлених сторонах – було визначено очікування відповідних сторін від продукту.

Розробники програмного забезпечення бажають знизити витрати на усунення архітектурних недоліків та невідповідностей, оскільки це, по перше – не завжди можливо чи доцільно (інколи простіше розробити нове ПЗ з нуля), по друге – усунення дефектів програмного забезпечення на рівні архітектури є надзвичайно дорогим та складним на релізній чи перед релізній стадії. Запропоноване рішення дозволяє задовольнити всі наведені вище вимоги зацікавлених сторін краще, ніж доступні на сьогоднішній день аналоги. Отже, основною унікальною пропозицією є розроблене програмне рішення для автоматизації процесу специфікації варіантів використання на основі їх атрибутів.

5.7. Доходи та витрати

Основним джерелом доходу є продаж поновлюваних ліцензій на використання програмного забезпечення, а також надання розширеної технічної підтримки. Під поновлюваністю мається на увазі надання ліцензії на тимчасове користування послугою або програмним продуктом, тобто використовується так звана модель підписки. Платна розширена технічна підтримка, у порівнянні зі стандартною, скорочує терміни надання відповіді на запити користувача та усунення дефектів. Іншими додатковими послугами є адаптація програмного забезпечення та розширення функціональності в залежності від потреби користувача.

Основні статті витрат складатимуть:

- проведення науково-дослідних та дослідно-конструкторських робіт;

- оплата праці персоналу;
- утримання робочих місць персоналу та виробничої інфраструктури;
- придбання обладнання, ліцензій програмного забезпечення для розробки;
- податкові витрати.

Витрати на реалізацію проекту та прогнозовані прибутки за перший рік діяльності стартап-проекту зведені у табл. 13.

Таблиця 13

Витрати на реалізацію проекту, тис. доларів США

Найменування витрат	**Місяці**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
Загальні витрати	7	4	4	4	4	4	4
ЗП	10	10	10	10	10	10	10
Підсумок витрат	17	14	14	14	14	14	14
Заплановані прибутки	0	0	0	0	0	0	0
Результат (без оподаткування)	–17	–14	–14	–14	–14	–14	–14
Найменування витрат	**Місяці**					**Загальні результати**	
	8	**9**	**10**	**11**	**12**		
Загальні витрати	4	4	4	4	4	**51**	
ЗП	10	10	10	10	10	**112**	
Підсумок витрат	14	14	14	14	14	**163**	
Заплановані прибутки	40	45	50	50	55	**240**	
Результат (без оподаткування)	26	31	36	36	41	**69**	

5.8. Бізнес-модель

В даному підрозділі слід узагальнити попередні підрозділи у вигляді бізнес-моделі, побудованої за шаблоном lean canvas.

Споживачі: компанії, що виконують розроблення програмного забезпечення.

Проблема: надмірна кількість паперової та ручних розрахунків при специфікації варіантів використання.

Рішення: програмне рішення для автоматизації процесу специфікування варіантів використання на основі їх атрибутів.

Унікальна ціннісна пропозиція: автоматизація вибору рівня деталізації, підвищення гнучкості даного алгоритму та автоматичне оповіщення при внесенні змін до специфікації.

Потоки доходів: доходи від продажу ліцензій програмного забезпечення; доходи від підтримки програмного забезпечення.

Структура витрат: проведення науково-дослідних та дослідно-конструкторських робіт; оплата праці персоналу; утримання робочих місць персоналу та виробничої інфраструктури; придбання обладнання, ліцензій програмного забезпечення для розробки; податкові витрати.

Канали: відділи інтеграції компаній, які займаються розробленням програмного забезпечення.

Ключові метрики: кількість проданих ліцензій.

Прихована перевага: веб-орієнтованість, один примірник на одну компанію підвищує захищеність даних.

У зведеному вигляді описана бізнес-модель наведена у табл 14. На основі цих даних можливо стверджувати про перспективність реалізації програмного продукту на основі розроблюваного в даній дисертації методу ефективного відтворення звукового поля в приміщеннях.

Про комерційну життєздатність даного стартап-проекту можливо стверджувати не зважаючи на те, що проведений аналіз не враховує всіх

факторів та ризиків, пов’язаних, наприклад, із податковою політикою країни, в якій ведеться бізнес.

Таблиця 14

Канва бізнес-моделі

<table>
<tr>
<td rowspan="2">Проблема
надмірна кількість паперової та ручних розрахунків при специфікації варіантів використання</td>
<td>Рішення
програмне рішення для автоматизації процесу специфікування варіантів використання на основі їх атрибутів</td>
<td rowspan="2">Унікальна ціннісна пропозиція
автоматизація вибору рівня деталізації, підвищення гнучкості даного алгоритму та автоматичне оповіщення при внесенні змін до специфікації</td>
<td>Приховані переваги
веб-оріентованість
один примірник на одну компанію підвищує захищеність даних</td>
<td rowspan="2">Споживачі
компанії, що виконують розроблення програмного забезпечення</td>
</tr>
<tr>
<td>Ключові показники
кількість проданих ліцензій</td>
<td>Канали
відділи інтеграції компаній, які виконують розроблення програмного забезпечення</td>
</tr>
<tr>
<td colspan="3">Структура витрат
проведення науково-дослідних та дослідно-конструкторських робіт
оплата праці персоналу
утримання робочих місць персоналу та виробничої інфраструктури
придбання обладнання, ліцензій програмного забезпечення для розробки
податкові витрати</td>
<td colspan="2">Потоки доходів
доходи від продажу ліцензій програмного забезпечення
доходи від підтримки програмного забезпечення</td>
</tr>
</table>

5.9. Висновки до розділу 5

У даному розділі було проведено аналіз поточної ситуації у сфері специфікації варіантів використання, виявлено наявні проблеми та підсумовано їх у відповідному дереві проблем. Крім проблем було виділено основні зацікавлені сторони у вирішенні їх існуючих потреб, ступінь впливу даних сторін на вирішення проблем. Як наслідок було запропоновано комерційне рішення з конкурентними перевагами, що задовольняє інтереси зацікавлених осіб, та виділено унікальну ціннісну пропозицію запропонованого продукту. Було проведено аналіз майбутніх клієнтів, досліджено сегменти ринку споживання. Це дозволило спрогнозувати потенційні доходи та витрати на реалізацію продукту. У результаті була описана бізнес-модель, що обґрунтовує доцільність реалізації даного продукту та прогнозує його потенційну окупність та прибутковість.

ВИСНОВКИ

В першому розділі було проведено огляд та аналіз існуючих шаблонів текстових специфікацій варіантів використання. Показано, що ці шаблони на різних рівнях деталізації описують ВВ. Сучасні програмні засоби, які використовуються для роботи над проектами, пропонують використання певного шаблону, який є прийнятим для реалізації в цьому ПЗ і не надає можливість використання інших шаблонів, або зміни рівня деталізації відповідної специфікації.

В другому розділі було проведено аналіз атрибутів, які використовуються в різних методах пріоретизації вимог та розроблено метод вибору рівня деталізації специфікації варіанту використання. Метод MoSCoW та модель Кано в їх класичному вигляді є сумісними з розробленим алгоритмом, проте ефективніше використовувати власні атрибути чи комбінацію з атрибутів даних методів, оскільки це розбиває задачу визначення пріоритету на кілька простіших – обрати значення атрибутів. Крім цього було обрано такі шаблони специфікацій, які дозволяють легко нарощувати рівень деталізації та спростити супровід за рахунок їх модульного подання. Описана в цьому розділі процедура автоматизованого створення специфікації ВВ базується на сформульованому методі вибору рівня деталізації та використовує модульне подання специфікації.

В третьому розділі було описано основні інструменти розроблення, детально проаналізовано архітектуру, особливості реалізації та функціональність програмного продукту, що автоматизує створення специфікацій варіантів використання на основі їх атрибутів. Також продемонстровано перехід варіанта використання від реєстру використання та нарощування його деталізації до четвертого рівня деталізації, який є найбільш повним і точним. Цьому сприяє вдало підібраний набір шаблонів та модульність представлення специфікацій. При цьому специфікація розглядається не як монолітна сутність, а як набір певних компонентів, які

в свою чергу можна легко додавати, прибирати та змінювати. Крім того, продемонстровано адаптивність інтерфейсу відповідно до типу користувача та обраного рівня деталізації специфікації. Вибір рівня деталізації можливий як з допомогою способу, описаного у другому розділі, так і вручну.

В четвертому розділі було описано методику тестування та на яких етапах які типи тестування застосовувалися. Тобто на початку тестувалася структура бази даних на можливість повної реалізації розроблюваного програмного забезпечення. В процесі розроблення застосовувалося статичне тестування коду та рефакторинг, поки не було користувацького інтерфейсу, потім додатково проводилось тестування методом сірого ящика. У результаті проведених фінального етапу тестування не знайдено помилок в роботі програми. Також були сформовані обмеження та рекомендації відносно серверної частини. Вимоги до клієнтської частини є достатньо низькими. Крім цього було описано основні шляхи подальшого вдосконалення програмного продукту, що в майбутньому дозволить підвищити зручність використання та розширити сферу його застосування.

В останньому розділі було проведено аналіз поточної ситуації у сфері специфікації варіантів використання, виявлено наявні проблеми та підсумовано їх у відповідному дереві проблем. Крім проблем було виділено основні зацікавлені сторони у вирішенні їх існуючих потреб, ступінь впливу даних сторін на вирішення проблем. Як наслідок було запропоновано комерційне рішення з конкурентними перевагами, що задовольняє інтереси зацікавлених осіб, та виділено унікальну ціннісну пропозицію запропонованого продукту. Було проведено аналіз майбутніх клієнтів, досліджено сегменти ринку споживання. Це дозволило спрогнозувати потенційні доходи та витрати на реалізацію продукту. У результаті була описана бізнес-модель, що обґрунтовує доцільність реалізації даного продукту та прогнозує його потенційну окупність та прибутковість.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Alistair Cockburn, Writing Effective Use Cases [Text] / Alistair Cockburn. — 1st edition. — Addison-Westley, 2001. — 264 p.
2. Karl Wiegers, software requirements [Text] / Karl Wiegers, Joy Beatty. — 3rd edition. — Twist Creative, Seattle, 2013. — 672 p.
3. Dean Leffingwell, Managing Software Requirements: A Use Case Approach [Text] / Dean Leffingwell, Don Widrig. — 2nd edition. — Addison-Westley, 2003. — 544 p.
4. Allen H. Dutoit, Rationale-based use case specification [Text] / Allen H. Dutoit, Raymond McCall, Ivan Mistrik, Barbara Paech. — Springer-Verlag London Limited, 2002. — 430 p.
5. Dusan Savic, Language for use case specification [Text] / Dusan Savic, Ilija Antovic, Sinisa Vlajic, Vojislav Stanojevic, Milos Mili. — 1st edition. — Limerick, Ireland, 2011 IEEE 34th Software Engineering Workshop, 2011.
6. MySQL [Електронний ресурс] — 2019. — Режим доступу: https://ru.wikipedia.org/wiki/MySQL — Дата доступу: 30.11.2019 — Назва з екрану.
7. СКБД MySQL [Електронний ресурс] — 2019. — Режим доступу: http://www.webmasterwiki.ru/mysql — Дата доступу: 30.11.2019 — Назва з екрану.
8. MariaDB [Електронний ресурс] — 2019. — Режим доступу: https://mariadb.com/kb/ru/mariadb–vs–mysql–features/ — Дата доступу: 30.11.2019 — Назва з екрану.
9. ORM [Електронний ресурс] — 2019. — Режим доступу: https://habr.com/post/237889/ — Дата доступу: 30.11.2019 — Назва з екрану.

10. Django [Електронний ресурс] — 2019. — Режим доступу: http://artkiev.com/blog/django-framework.htm — Дата доступу: 30.11.2019 — Назва з екрану.
11. Django документація [Електронний ресурс] — 2019. — Режим доступу: https://djbook.ru/ — Дата доступу: 30.11.2019 — Назва з екрану.
12. Python [Електронний ресурс] — 2019. — Режим доступу: https://pythonworld.ru/ — Дата доступу: 30.11.2019 — Назва з екрану.
13. MVC/MTV[Електронний ресурс] — 2019. — Режим доступу: https://djbook.ru/ch05s02.html — Дата доступу: 30.11.2019 — Назва з екрану.
14. URL [Електронний ресурс] — 2019. — Режим доступу: http://promo-sa.ru/seo-terms/URL-adress — Дата доступу: 30.11.2019 — Назва з екрану.
15. C++ [Електронний ресурс] — 2019. — Режим доступу: http://cppstudio.com/post/213/ — Дата доступу: 30.11.2019 — Назва з екрану.
16. Bootstrap [Електронний ресурс] — 2019. — Режим доступу: http://blogwork.ru/chto-takoe-bootstrap/ — Дата доступу: 30.11.2019 — Назва з екрану.
17. Аутентифікація користувачів в Django [Електронний ресурс] — 2019. — Режим доступу: https://djbook.ru/rel2.1/topics/auth.html — Дата доступу: 30.11.2019 — Назва з екрану.
18. Gray Box Testing [Електронний ресурс] — 2019. — Режим доступу: http://ru.qatestlab.com/knowledge-center/qa-testing-materials/the-principles-of-gray-box-testing/ — Дата доступу: 30.11.2019 — Назва з екрану.

ДОДАТКИ

Додаток 1

Лістинг програми

Лістинг 1. models.py

```
from django.db import models
from math import *
from spec.choices import *

class Project(models.Model):
    name = models.CharField(max_length=250)

class Attribute(models.Model):
    weigh = models.FloatField(default=1.0)
    name = models.CharField(max_length=250)
    gradation_level = models.IntegerField(choices=GRADATION_CHOICES,
default=4)
    project_link = models.ForeignKey(Project, on_delete=models.CASCADE)

class Actor(models.Model):
    name = models.CharField(max_length=250)
    project_link = models.ForeignKey(Project, on_delete=models.CASCADE)

class UseCase(models.Model):
    name = models.CharField(max_length=250)
    recommended_level = models.IntegerField(default=0)
    actual_level = models.IntegerField(default=0)
    log = models.TextField(blank=True, null=True)
    project_link = models.ForeignKey(Project, on_delete=models.CASCADE)
    actor_link = models.ForeignKey(Actor, blank=True, null=True,
on_delete=models.SET_NULL)
    description = models.TextField(blank=True, null=True)
    purpose = models.TextField(blank=True, null=True)
    main_flow_free_format = models.TextField(blank=True, null=True)

class Condition(models.Model):
    type = models.IntegerField(choices=TYPE_CHOICES, default=1)
    description = models.TextField()
    use_case_link = models.ForeignKey(UseCase, on_delete=models.CASCADE)

class Step(models.Model):
    number = models.CharField(max_length=20)
    text = models.TextField()
    use_case_link = models.ForeignKey(UseCase, blank=True, null=True,
on_delete=models.CASCADE)

class ExceptionOrAlternative(models.Model):
    code = models.CharField(max_length=50)
    description = models.TextField()
    name = models.CharField(max_length=250, blank=True, null=True)

class ManyToManyAltStep(models.Model):
    step_link = models.ForeignKey(Step, on_delete=models.CASCADE)
    exception_or_alternative_link =
models.ForeignKey(ExceptionOrAlternative, on_delete=models.CASCADE)
    is_step_to_alt = models.BooleanField()

class Extension(models.Model):
    step_link = models.ForeignKey(Step, on_delete=models.CASCADE)
    use_case_link = models.ForeignKey(UseCase, on_delete=models.CASCADE)
    description = models.TextField()
    code = models.CharField(max_length=50, default='Ext')
```

```
class AttributeValue(models.Model):
    attribute_link = models.ForeignKey(Attribute, on_delete=models.CASCADE)
    use_case_link = models.ForeignKey(UseCase, on_delete=models.CASCADE)
    value = models.IntegerField(default=1)
class Trigger(models.Model):
    name = models.CharField(max_length=250)
    use_case_link = models.ForeignKey(UseCase, on_delete=models.CASCADE)

def get_trigger_by_id(trigger_id):
    return Trigger.objects.get(id=trigger_id)

def get_trigger_list(use_case_id):
    return Trigger.objects.filter(use_case_link_id=use_case_id)

def add_trigger(name, use_case_id):
    Trigger.objects.create(name=name, use_case_link_id=use_case_id)

def delete_trigger(trigger_id):
    trigger = get_trigger_by_id(trigger_id)
    trigger.delete()

def edit_trigger(trigger_id, name):
    trigger = get_trigger_by_id(trigger_id)
    trigger.name = name
    trigger.save()

def get_actor_by_id(actor_id):
    return Actor.objects.get(id=actor_id)

def get_actor_by_id_safe(actor_id):
    actor = Actor.objects.filter(id=actor_id)[:1]
    if not actor:
        return None
    return actor[0]

def get_attribute_value(attribute_id, use_case_id):
    return AttributeValue.objects.get(attribute_link_id=attribute_id,
use_case_link_id=use_case_id)

def get_attribute_value_list_uc(use_case_id):
    use_case = get_use_case_by_id(use_case_id)
    attribute_list = get_attribute_list(use_case.project_link_id)
    attribute_values = []
    for attribute in attribute_list:
        tmp = get_attribute_value(attribute.id, use_case.id)
        attribute_values.append(tmp)
    return attribute_values

def set_attribute_value(attribute_id, use_case_id, value):
    attribute_value = get_attribute_value(attribute_id, use_case_id)
    attribute_value.value = value
    attribute_value.save()

def get_project_by_id(project_id):
    return Project.objects.get(id=project_id)

def get_project_list():
    return Project.objects.all()

def get_attribute_by_id(attribute_id):
    return Attribute.objects.get(id=attribute_id)
```

```
def get_use_case_by_id(use_case_id):
    return UseCase.objects.get(id=use_case_id)

def get_condition_by_id(condition_id):
    return Condition.objects.get(id=condition_id)

def get_step_by_id(step_id):
    return Step.objects.get(id=step_id)

def get_exception_or_alternative_by_id(exception_or_alternative_id):
    return
ExceptionOrAlternative.objects.get(id=exception_or_alternative_id)

def get_extension_by_id(extension_id):
    return Extension.objects.get(id=extension_id)

def add_extension(step_id, use_case_id, code, description):
    Extension.objects.create(step_link_id=step_id,
use_case_link_id=use_case_id, code=code, description=description)


def delete_extension(extension_id):
    extension = get_extension_by_id(extension_id)
    extension.delete()

def delete_extension_with_find(use_case_id, step_id):
    extensions = Extension.objects.filter(use_case_link_id=use_case_id,
step_link_id=step_id)
    for extension in extensions:
        extension.delete()

def get_attribute_list(project_id):
    return
Attribute.objects.filter(project_link_id=project_id).order_by('name')

def get_use_case_list_with_attribute_values(project_id):
    use_case_list = get_use_case_list(project_id)
    use_case_list_with_attribute_values = []
    for use_case in use_case_list:
        tmp = {}
        values = get_attribute_value_list_uc(use_case.id)
        tmp['use_case'] = use_case
        tmp['actor'] = get_actor_by_id_safe(use_case.actor_link_id)
        tmp['attribute_values'] = values
        use_case_list_with_attribute_values.append(tmp)
    return use_case_list_with_attribute_values

def get_use_case_list_with_actors(project_id):
    use_case_list =
UseCase.objects.filter(project_link_id=project_id).order_by('actual_level')
    use_case_list_with_actors = []
    for use_case in use_case_list:
        tmp = {}
        tmp['use_case'] = use_case
        tmp['actor'] = get_actor_by_id_safe(use_case.actor_link_id)
        use_case_list_with_actors.append(tmp)
    return use_case_list_with_actors

def get_use_case_list(project_id):
    return
```

```
UseCase.objects.filter(project_link_id=project_id).order_by('name')


def get_condition_list(use_case_id):
    return Condition.objects.filter(use_case_link_id=use_case_id)

def get_step_list_from_use_case(use_case_id):
    return
Step.objects.filter(use_case_link_id=use_case_id).order_by('number')

def get_step_number_from_use_case(use_case_id):
    return Step.objects.filter(use_case_link_id=use_case_id).count()

def get_step_list_from_alt(alt_id):
    records =
ManyToManyAltStep.objects.filter(exception_or_alternative_link_id=alt_id,
is_step_to_alt=False)\
        .values_list('step_link_id', flat=True)
    return Step.objects.filter(id__in=records).order_by('number')

def get_step_number_from_alt(alt_id):
    records =
ManyToManyAltStep.objects.filter(exception_or_alternative_link_id=alt_id,
is_step_to_alt=False) \
        .values_list('step_link_id', flat=True)
    return Step.objects.filter(id__in=records).count()

def get_alt_list_from_step(step_id):
    records = ManyToManyAltStep.objects.filter(step_link_id=step_id,
is_step_to_alt=True)\
        .values_list('exception_or_alternative_link_id', flat=True)
    return ExceptionOrAlternative.objects.filter(id__in=records)

def get_parent_step_from_alt(alt_id):
    record =
ManyToManyAltStep.objects.filter(exception_or_alternative_link_id=alt_id,
is_step_to_alt=True)[:1]
    return get_step_by_id((record[0]).step_link_id)

def get_parent_alt_from_step(step_id):
    record = ManyToManyAltStep.objects.filter(step_link_id=step_id,
is_step_to_alt=False)[:1]
    return
get_exception_or_alternative_by_id((record[0]).exception_or_alternative_lin
k_id)

def get_extension_list_by_step(step_id):
    return Extension.objects.filter(step_link_id=step_id)

def get_extension_list_by_use_case(use_case_id):
    return Extension.objects.filter(use_case_link_id=use_case_id)

def add_project(name):
    Project.objects.create(name=name)

def add_attribute(project_id, name, weigh, gradation_level):
    attribute = Attribute.objects.create(
        name=name, weigh=weigh, gradation_level=gradation_level,
project_link_id=project_id)
    use_case_list = get_use_case_list(project_id)
    for use_case in use_case_list:
```

```
        AttributeValue.objects.create(value=1,
use_case_link_id=use_case.id, attribute_link_id=attribute.id)

def get_actor_list(project_id):
    return
Actor.objects.filter(project_link_id=project_id).order_by('name')

def add_actor(project_id, name):
    Actor.objects.create(name=name, project_link_id=project_id)

def edit_actor(actor_id, actor_name):
    actor = get_actor_by_id(actor_id)
    actor.name = actor_name
    actor.save()

def delete_actor(actor_id):
    actor = get_actor_by_id(actor_id)
    actor.delete()

def add_use_case(name, purpose, actor_id, project_id):
    use_case = UseCase.objects.create(name=name,
project_link_id=project_id, actor_link_id=actor_id,
                                      actual_level=1, recommended_level=1,
purpose=purpose)
    attribute_list = get_attribute_list(project_id)
    for attribute in attribute_list:
        AttributeValue.objects.create(value=1,
use_case_link_id=use_case.id, attribute_link_id=attribute.id)

def add_condition(condition_type, description, use_case_id):
    Condition.objects.create(type=condition_type, description=description,
use_case_link_id=use_case_id)

def add_step(number, text, use_case_id):
    Step.objects.create(number=number, text=text,
use_case_link_id=use_case_id)

def add_step_from_alt(number, text, alt_id):
    step = Step.objects.create(number=number, text=text)
    ManyToManyAltStep.objects.create(
        step_link_id=step.id, exception_or_alternative_link_id=alt_id,
is_step_to_alt=False)

def add_exception_or_alternative(code, description, name, step_id):
    alt = ExceptionOrAlternative.objects.create(code=code,
description=description, name=name)
    ManyToManyAltStep.objects.create(step_link_id=step_id,
exception_or_alternative_link_id=alt.id, is_step_to_alt=True)

def edit_project(project_id, project_name):
    project = get_project_by_id(project_id)
    project.name = project_name
    project.save()

def delete_project(project_id):
    project = get_project_by_id(project_id)
    project.delete()

def edit_alternative(alternative_id, code, description, name):
    alternative = get_exception_or_alternative_by_id(alternative_id)
    alternative.code = code
```

```
    alternative.description = description
    alternative.name = name
    alternative.save()
def edit_attribute(attribute_id, name, weigh):
    attribute = get_attribute_by_id(attribute_id)
    attribute.name = name
    attribute.weigh = weigh
    attribute.save()

def delete_attribute(attribute_id):
    attribute = get_attribute_by_id(attribute_id)
    attribute.delete()

def edit_condition(condition_id, type, description):
    condition = get_condition_by_id(condition_id)
    condition.type = type
    condition.description = description
    condition.save()

def delete_condition(condition_id):
    condition = get_condition_by_id(condition_id)
    condition.delete()

def delete_alternative(alternative_id):
    alternative = get_exception_or_alternative_by_id(alternative_id)
    alt_step_list = get_step_list_from_alt(alternative.id)
    for alt_step in alt_step_list:
        delete_step(alt_step.id)
    alternative.delete()

def delete_step(step_id):
    step = get_step_by_id(step_id)
    alt_list = get_alt_list_from_step(step.id)
    for alt in alt_list:
        delete_alternative(alt.id)
    step.delete()

def step_list_with_alts(steps_qset):
    step_list_with_alt = []
    for step in steps_qset:
        tmp = {}
        alt_list = get_alt_list_from_step(step.id)
        ext_list = get_extension_list_by_step(step.id)
        tmp['step'] = step
        tmp['alt_list'] = alt_list
        tmp['ext_list'] = ext_list
        step_list_with_alt.append(tmp)
    return step_list_with_alt

def save_step_formset_with_ids(formset, id_list):
    for form, step_id in zip(formset, id_list):
        step = get_step_by_id(step_id['id'])
        step.number = form.cleaned_data['number']
        step.text = form.cleaned_data['text']
        step.save()

def save_attribute_value_formset(formset, use_case_id):
    for form in formset:
        attribute_value =
get_attribute_value(form.cleaned_data['attribute_id'], use_case_id)
        attribute_value.value = form.cleaned_data['value']
```

```
        attribute_value.save()
def edit_use_case(use_case_id, name, description, purpose, actor):
    use_case = get_use_case_by_id(use_case_id)
    use_case.name = name
    use_case.description = description
    use_case.purpose = purpose
    use_case.actor_link_id = actor
    use_case.save()

def delete_use_case(use_case_id):
    use_case = get_use_case_by_id(use_case_id)
    use_case.delete()

def set_recommended_level_use_case(use_case_id):
    use_case = get_use_case_by_id(use_case_id)
    attribute_list = get_attribute_list(use_case.project_link_id)
    numerator = 0
    denominator = 0
    for attribute in attribute_list:
        attribute_value = get_attribute_value(attribute.id, use_case.id)
        numerator += (attribute.weigh * attribute_value.value * 4 /
attribute.gradation_level)
        denominator += attribute.weigh
    recommended_level = round(numerator / denominator)
    use_case.recommended_level = recommended_level
    use_case.actual_level = recommended_level
    use_case.save()

def set_actual_level_use_case(use_case_id, actual_level):
    use_case = get_use_case_by_id(use_case_id)
    use_case.actual_level = actual_level
    use_case.save()

def get_unique_alternative_list(step_id):
    step = get_step_by_id(step_id)
    if step.use_case_link_id is None:
        alternative = get_parent_alt_from_step(step.id)
        step_list = get_step_list_from_alt(alternative.id)
    else:
        use_case = get_use_case_by_id(step.use_case_link_id)
        step_list = get_step_list_from_use_case(use_case.id)
    step_list_a = step_list_with_alts(step_list)
    alt_ids = []
    for step in step_list_a:
        alt_list = step['alt_list']
        for alt in alt_list:
            if alt.id not in alt_ids:
                alt_ids.append(alt.id)
    alt_list = ExceptionOrAlternative.objects.filter(id__in=alt_ids)
    return alt_list

def link_alternative(step_id, alternative_id):
    step = get_step_by_id(step_id)
    alternative = get_exception_or_alternative_by_id(alternative_id)
    ManyToManyAltStep.objects.create(
        step_link_id=step.id,
exception_or_alternative_link_id=alternative.id, is_step_to_alt=True)

def unlink_alternative(step_id, alternative_id):
    step = get_step_by_id(step_id)
    alternative = get_exception_or_alternative_by_id(alternative_id)
```

```
    records = ManyToManyAltStep.objects.filter(
        step_link_id=step.id,
exception_or_alternative_link_id=alternative.id, is_step_to_alt=True)
    for record in records:
        record.delete()

def step_have_alternatives(step_id):
    alt_list = get_alt_list_from_step(step_id)
    have_alternatives = False
    for alt in alt_list:
        have_alternatives = True
    return have_alternatives

def parent_have_alternatives(step_id):
    have_alternatives = False
    alt_list = get_unique_alternative_list(step_id)
    for alt in alt_list:
        have_alternatives = True
    return have_alternatives

def edit_extension(extension_id, code, description):
    extension = get_extension_by_id(extension_id)
    extension.code = code
    extension.description = description
    extension.save()
```

Лістинг 2. views.py

```
from django.shortcuts import render, redirect
from . import models
from .forms import *
from django.contrib import auth
from django.contrib.auth.decorators import login_required

@login_required
def project_list(request):
    context = {}
    context['project_list'] = models.get_project_list()
    return render(request, 'spec/project_list.html', context)

@login_required
def attribute_list(request, project_id):
    project = models.get_project_by_id(project_id)
    context = {}
    context['attribute_list'] = models.get_attribute_list(project_id)
    context['project'] = project
    return render(request, 'spec/attribute_list.html', context)

@login_required
def use_case_list(request, project_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    project = models.get_project_by_id(project_id)
    attribute_list = models.get_attribute_list(project.id)
    context = {}
    use_case_list_with_val =
models.get_use_case_list_with_attribute_values(project_id)
```

```
    context['use_case_list_with_val'] = use_case_list_with_val
    context['project'] = project
    context['attribute_list'] = attribute_list
    return render(request, 'spec/use_case_list.html', context)

@login_required
def add_trigger(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    if request.method == 'POST':
        form = EditNameForm(request.POST)
        if form.is_valid():
            models.add_trigger(form.cleaned_data['name'], use_case.id)
            return redirect('/spec/use_case_description/%i/' % use_case.id)
    else:
        form = EditNameForm()
        context = {'form': form, 'use_case': use_case}
        return render(request, 'spec/add_trigger.html', context)

@login_required
def delete_trigger(request, trigger_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    trigger = models.get_trigger_by_id(trigger_id)
    use_case = models.get_use_case_by_id(trigger.use_case_link_id)
    if request.method == 'POST':
        models.delete_trigger(trigger_id)
        return redirect('/spec/use_case_description/%i/' % use_case.id)
    else:
        trigger = models.get_trigger_by_id(trigger_id)
        context = {'trigger': trigger}
        return render(request, 'spec/delete_trigger.html', context)

@login_required
def edit_trigger(request, trigger_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    trigger = models.get_trigger_by_id(trigger_id)
    use_case = models.get_use_case_by_id(trigger.use_case_link_id)
    if request.method == 'POST':
        form = EditNameForm(request.POST)
        if form.is_valid():
            models.edit_trigger(trigger_id, form.cleaned_data['name'])
            return redirect('/spec/use_case_description/%i/' % use_case.id)
    else:
        form = EditNameForm(initial={'name': trigger.name})
        context = {'form': form, 'trigger': trigger, 'use_case': use_case}
        return render(request, 'spec/edit_trigger.html', context)

@login_required
def use_case_register(request, project_id):
    project = models.get_project_by_id(project_id)
    context = {}
    use_case_list_with_actors =
models.get_use_case_list_with_actors(project_id)
    context['use_case_list_with_actors'] = use_case_list_with_actors
    context['project'] = project
```

```
    return render(request, 'spec/use_case_register.html', context)

@login_required
def use_case_description(request, use_case_id):
    use_case = models.get_use_case_by_id(use_case_id)
    trigger_list = models.get_trigger_list(use_case.id)
    context = {}
    context['use_case'] = use_case
    context['trigger_list'] = trigger_list
    context['actor'] = models.get_actor_by_id_safe(use_case.actor_link_id)
    context['project'] = models.get_project_by_id(use_case.project_link_id)
    step_list = models.get_step_list_from_use_case(use_case.id)
    step_list_with_alt = models.step_list_with_alts(step_list)
    context['step_list'] = step_list_with_alt
    conditions = models.get_condition_list(use_case.id)
    pre = []
    post = []
    spec = []
    for condition in conditions:
        if 1 == condition.type:
            pre.append(condition)
        elif 2 == condition.type:
            post.append(condition)
        else:
            spec.append(condition)
    context['pre_conditions'] = pre
    context['post_conditions'] = post
    context['spec_conditions'] = spec
    return render(request, 'spec/use_case_description.html', context)

@login_required
def edit_free_format_description(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    if request.method == 'POST':
        form = EditTextForm(request.POST)
        if form.is_valid():
            use_case.main_flow_free_format = form.cleaned_data['text']
            use_case.save()
            return redirect('/spec/use_case_description/%i/' % use_case.id)
    else:
        form = EditTextForm(initial={'text':
use_case.main_flow_free_format})
        project = models.get_project_by_id(use_case.project_link_id)
        context = {'form': form, 'use_case': use_case, 'project': project}
        return render(request, 'spec/edit_free_format_description.html',
context)

@login_required
def choose_level_of_detalisation(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    attributeFormSet = formset_factory(form=EditAttributeValueForm,
extra=0)
    attribute_list = models.get_attribute_list(use_case.project_link_id)
    initial_data = [{'attribute_name': q.name, 'attribute_id': q.id} for q
in attribute_list]
```

```
    formset = attributeFormSet(initial=initial_data)
    for form, attribute in zip(formset, attribute_list):
        if 5 == attribute.gradation_level:
            form.fields['value'].choices = CHOICES_5
        elif 4 == attribute.gradation_level:
            form.fields['value'].choices = CHOICES_4
        else:
            form.fields['value'].choices = CHOICES_3
    if request.method == 'POST':
        formset = attributeFormSet(request.POST)
        for form, attribute in zip(formset, attribute_list):
            if 5 == attribute.gradation_level:
                form.fields['value'].choices = CHOICES_5
            elif 4 == attribute.gradation_level:
                form.fields['value'].choices = CHOICES_4
            else:
                form.fields['value'].choices = CHOICES_3
        if formset.is_valid():
            models.save_attribute_value_formset(formset, use_case.id)
            models.set_recommended_level_use_case(use_case.id)
        return redirect('/spec/use_case_list/%i/' %
use_case.project_link_id)
    context = {'formset': formset, 'use_case': use_case, }
    return render(request, 'spec/choose_level_of_detalisation.html',
context)

@login_required
def alt_description(request, alt_id):
    alt = models.get_exception_or_alternative_by_id(alt_id)
    context = {}
    context['alternative'] = alt
    step_list = models.get_step_list_from_alt(alt.id)
    step_list_with_alt = models.step_list_with_alts(step_list)
    context['step_list'] = step_list_with_alt
    alternative_list = []
    parent_step = models.get_parent_step_from_alt(alt.id)
    is_use_case_back = 1
    use_case_id = parent_step.use_case_link_id
    while parent_step.use_case_link_id is None:
        is_use_case_back = 0
        tmp = models.get_parent_alt_from_step(parent_step.id)
        alternative_list.append(tmp)
        parent_step = models.get_parent_step_from_alt(tmp.id)
        use_case_id = parent_step.use_case_link_id
    context['is_use_case_back'] = is_use_case_back
    use_case = models.get_use_case_by_id(use_case_id)
    context['use_case'] = use_case
    if is_use_case_back == 1:
        context['back_id'] = use_case.id
    else:
        back = alternative_list[0]
        context['back_id'] = back.id
    alternative_list.reverse()
    context['nav_list'] = alternative_list
    context['project'] = models.get_project_by_id(use_case.project_link_id)
    return render(request, 'spec/alt_description.html', context)

@login_required
def add_project(request):
    user = auth.get_user(request)
    if not user.is_staff:
```

```
        return redirect('/spec/permission_error/')
    if request.method == 'POST':
        form = EditNameForm(request.POST)
        if form.is_valid():
            models.add_project(form.cleaned_data['name'])
            return redirect('/spec/project_list/')
    else:
        form = EditNameForm()
        context = {'form': form}
        return render(request, 'spec/add_project.html', context)

@login_required
def edit_project(request, project_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    if request.method == 'POST':
        form = EditNameForm(request.POST)
        if form.is_valid():
            models.edit_project(project_id, form.cleaned_data['name'])
            return redirect('/spec/project_list/')
    else:
        project = models.get_project_by_id(project_id)
        form = EditNameForm(initial={'name': project.name})
        context = {'form': form, 'project': project}
        return render(request, 'spec/edit_project.html', context)

@login_required
def delete_project(request, project_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    if request.method == 'POST':
        models.delete_project(project_id)
        return redirect('/spec/project_list/')
    else:
        project = models.get_project_by_id(project_id)
        context = {'project': project}
        return render(request, 'spec/delete_project.html', context)

@login_required
def set_level_of_detalisation(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    if request.method == 'POST':
        form = EditLevelOfDetalisation(request.POST)
        if form.is_valid():
            models.set_actual_level_use_case(use_case.id,
form.cleaned_data['level_of_detalisation'])
            return redirect('/spec/use_case_list/%i/' %
use_case.project_link_id)
    else:
        form = EditLevelOfDetalisation()
        context = {'form': form, 'use_case': use_case}
        return render(request, 'spec/set_level_of_detalisation.html',
context)
@login_required
def edit_alternative(request, alternative_id):
    user = auth.get_user(request)
```

```
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    alternative = models.get_exception_or_alternative_by_id(alternative_id)
    parent_step = models.get_parent_step_from_alt(alternative_id)
    if request.method == 'POST':
        form = EditAltForm(request.POST)
        if form.is_valid():
            models.edit_alternative(alternative_id,
                                    form.cleaned_data['code'],
                                    form.cleaned_data['description'],
                                    form.cleaned_data['name'])
            return redirect('/spec/alt_description/%i/' % alternative.id)
    else:
        if parent_step.use_case_link_id is None:
            parent_is_use_case = False
            parent_alternative =
models.get_parent_alt_from_step(parent_step.id)
            back_id = parent_alternative.id
        else:
            parent_is_use_case = True
            back_id = parent_step.use_case_link_id
        form = EditAltForm(initial={'code': alternative.code,
'description': alternative.description})
        context = {'form': form, 'alternative': alternative,
                   'parent_is_use_case': parent_is_use_case, 'back_id':
back_id}
        return render(request, 'spec/edit_alternative.html', context)

@login_required
def link_alternative(request, step_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    step = models.get_step_by_id(step_id)
    alt_list = models.get_unique_alternative_list(step.id)
    if request.method == 'POST':
        form = ChooseSomething(request.POST)
        form.fields['something'].choices = [(q.id, q.code) for q in
alt_list]
        if form.is_valid():
            models.link_alternative(step_id,
form.cleaned_data['something'])
            if step.use_case_link_id is None:
                parent_alt = models.get_parent_alt_from_step(step.id)
                return redirect('/spec/edit_step_list_alt/%i/' %
parent_alt.id)
            else:
                use_case = models.get_use_case_by_id(step.use_case_link_id)
                return redirect('/spec/edit_step_list_use_case/%i/' %
use_case.id)
    else:
        form = ChooseSomething()
        form.fields['something'].choices = [(q.id, q.code) for q in
alt_list]
        context = {'form': form, 'step': step}
        return render(request, 'spec/link_alternative.html', context)

@login_required
def unlink_alternative(request, step_id):
    user = auth.get_user(request)
    if not user.is_staff:
```

```
        return redirect('/spec/permission_error/')
    step = models.get_step_by_id(step_id)
    alt_list = models.get_alt_list_from_step(step.id)
    if request.method == 'POST':
        form = ChooseSomething(request.POST)
        form.fields['something'].choices = [(q.id, q.code) for q in
alt_list]
        if form.is_valid():
            models.unlink_alternative(step_id,
form.cleaned_data['something'])
            if step.use_case_link_id is None:
                parent_alt = models.get_parent_alt_from_step(step.id)
                return redirect('/spec/edit_step_list_alt/%i/' %
parent_alt.id)
            else:
                use_case = models.get_use_case_by_id(step.use_case_link_id)
                return redirect('/spec/edit_step_list_use_case/%i/' %
use_case.id)
    else:
        form = ChooseSomething()
        form.fields['something'].choices = [(q.id, q.code) for q in
alt_list]
        context = {'form': form, 'step': step}
        return render(request, 'spec/unlink_alternative.html', context)

@login_required
def delete_alternative(request, alternative_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    if request.method == 'POST':
        parent_step = models.get_parent_step_from_alt(alternative_id)
        if parent_step.use_case_link_id is not None:
            models.delete_alternative(alternative_id)
            return redirect('/spec/use_case_description/%i/' %
parent_step.use_case_link_id)
        else:
            parent_alternative =
models.get_parent_alt_from_step(parent_step.id)
            models.delete_alternative(alternative_id)
            return redirect('/spec/alt_description/%i/' %
parent_alternative.id)
    else:
        project = models.get_exception_or_alternative_by_id(alternative_id)
        context = {'project': project}
        return render(request, 'spec/delete_alternative.html', context)

@login_required
def add_attribute(request, project_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    project = models.get_project_by_id(project_id)
    if request.method == 'POST':
        form = AddAttributeForm(request.POST)
        if form.is_valid():
            models.add_attribute(project_id, form.cleaned_data['name'],
                                 form.cleaned_data['weigh'],
form.cleaned_data['gradation_level'])
            return redirect('/spec/attribute_list/%i/' % project.id)
    else:
```

```
        form = AddAttributeForm(initial={'weigh': 1})
        context = {'form': form, 'project': project}
        return render(request, 'spec/add_attribute.html', context)

@login_required
def edit_attribute(request, attribute_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    if request.method == 'POST':
        form = EditAttributeForm(request.POST)
        if form.is_valid():
            attribute = models.get_attribute_by_id(attribute_id)
            project_id = attribute.project_link_id
            models.edit_attribute(attribute_id, form.cleaned_data['name'],
form.cleaned_data['weigh'])
            return redirect('/spec/attribute_list/%i/' % project_id)
    else:
        attribute = models.get_attribute_by_id(attribute_id)
        project = models.get_project_by_id(attribute.project_link_id)
        form = EditAttributeForm(initial={'name': attribute.name, 'weigh':
attribute.weigh})
        context = {'form': form, 'attribute': attribute, 'project':
project}
        return render(request, 'spec/edit_attribute.html', context)

@login_required
def delete_attribute(request, attribute_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    attribute = models.get_attribute_by_id(attribute_id)
    if request.method == 'POST':
        project_id = attribute.project_link_id
        models.delete_attribute(attribute_id)
        return redirect('/spec/attribute_list/%i/' % project_id)
    else:
        context = {'attribute': attribute}
        return render(request, 'spec/delete_attribute.html', context)

@login_required
def actor_list(request, project_id):
    project = models.get_project_by_id(project_id)
    context = {}
    context['actor_list'] = models.get_actor_list(project_id)
    context['project'] = project
    return render(request, 'spec/actor_list.html', context)

@login_required
def add_actor(request, project_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    project = models.get_project_by_id(project_id)
    if request.method == 'POST':
        form = EditNameForm(request.POST)
        if form.is_valid():
            models.add_actor(project_id, form.cleaned_data['name'])
            return redirect('/spec/actor_list/%i/' % project.id)
    else:
        form = EditNameForm()
```

```
        context = {'form': form, 'project': project}
        return render(request, 'spec/add_actor.html', context)

@login_required
def edit_actor(request, actor_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    actor = models.get_actor_by_id(actor_id)
    if request.method == 'POST':
        form = EditNameForm(request.POST)
        if form.is_valid():
            project_id = actor.project_link_id
            models.edit_actor(actor_id, form.cleaned_data['name'])
            return redirect('/spec/actor_list/%i/' % project_id)
    else:
        project = models.get_project_by_id(actor.project_link_id)
        form = EditNameForm(initial={'name': actor.name})
        context = {'form': form, 'actor': actor, 'project': project}
        return render(request, 'spec/edit_actor.html', context)

@login_required
def delete_actor(request, actor_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    actor = models.get_actor_by_id(actor_id)
    if request.method == 'POST':
        project_id = actor.project_link_id
        models.delete_actor(actor.id)
        return redirect('/spec/actor_list/%i/' % project_id)
    else:
        context = {'actor': actor}
        return render(request, 'spec/delete_actor.html', context)

@login_required
def add_condition(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    if request.method == 'POST':
        form = EditConditionForm(request.POST)
        if form.is_valid():
            models.add_condition(form.cleaned_data['type'],
form.cleaned_data['description'], use_case.id)
            return redirect('/spec/use_case_description/%i/' % use_case.id)
    else:
        form = EditConditionForm()
        context = {'form': form, 'use_case': use_case}
        return render(request, 'spec/add_condition.html', context)

@login_required
def edit_condition(request, condition_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    if request.method == 'POST':
        form = EditConditionForm(request.POST)
        if form.is_valid():
            condition = models.get_condition_by_id(condition_id)
```

```
            use_case_id = condition.use_case_link_id
            models.edit_condition(condition_id, form.cleaned_data['type'],
form.cleaned_data['description'])
            return redirect('/spec/use_case_description/%i/' % use_case_id)
    else:
        condition = models.get_condition_by_id(condition_id)
        use_case = models.get_use_case_by_id(condition.use_case_link_id)
        project = models.get_project_by_id(use_case.project_link_id)
        form = EditConditionForm(initial={'type': condition.type,
'description': condition.description})
        context = {'form': form, 'condition': condition, 'use_case':
use_case, 'project': project}
        return render(request, 'spec/edit_condition.html', context)

@login_required
def delete_condition(request, condition_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    condition = models.get_condition_by_id(condition_id)
    if request.method == 'POST':
        use_case_id = condition.use_case_link_id
        models.delete_condition(condition_id)
        return redirect('/spec/use_case_description/%i/' % use_case_id)
    else:
        context = {'condition': condition}
        return render(request, 'spec/delete_condition.html', context)

@login_required
def add_use_case(request, project_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    project = models.get_project_by_id(project_id)
    if request.method == 'POST':
        form = AddUseCaseForm(request.POST)
        form.fields['actor'].choices = [(actor.id, actor.name) for actor in
Actor.objects.filter(
            project_link_id=project.id)]
        if form.is_valid():
            models.add_use_case(
                form.cleaned_data['name'], form.cleaned_data['purpose'],
form.cleaned_data['actor'], project.id)
        return redirect('/spec/use_case_list/%i/' % project.id)
    else:
        form = AddUseCaseForm()
        form.fields['actor'].choices = [(actor.id, actor.name) for actor in
Actor.objects.filter(
            project_link_id=project.id)]
        context = {'form': form, 'project': project}
        return render(request, 'spec/add_use_case.html', context)

@login_required
def edit_use_case(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    project = models.get_project_by_id(use_case.project_link_id)
    if request.method == 'POST':
        form = EditUseCaseForm(request.POST)
```

```
        form.fields['actor'].choices = [(actor.id, actor.name) for actor in
Actor.objects.filter(
            project_link_id=project.id)]
        if form.is_valid():
            models.edit_use_case(
                use_case_id, form.cleaned_data['name'],
form.cleaned_data['description'], form.cleaned_data['purpose'],
                form.cleaned_data['actor'])
            return redirect('/spec/use_case_description/%i/' % use_case.id)
    else:
        form = EditUseCaseForm(
            initial={'name': use_case.name, 'description':
use_case.description, 'purpose': use_case.purpose})
        form.fields['actor'].choices = [(actor.id, actor.name) for actor in
Actor.objects.filter(
            project_link_id=project.id)]
        context = {'form': form, 'use_case': use_case, 'project': project}
        return render(request, 'spec/edit_use_case.html', context)

@login_required
def delete_use_case(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    if request.method == 'POST':
        project_id = use_case.project_link_id
        models.delete_use_case(use_case_id)
        return redirect('/spec/use_case_list/%i/' % project_id)
    else:
        context = {'use_case': use_case}
        return render(request, 'spec/delete_use_case.html', context)

@login_required
def edit_step_list_use_case(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    project = models.get_project_by_id(use_case.project_link_id)
    StepFormSet = formset_factory(form=EditStepForm, extra=0)
    qset = models.get_step_list_from_use_case(use_case.id)
    step_list_with_alt = models.step_list_with_alts(qset)
    initial_data = [{'text': q.text, 'number': q.number} for q in qset]
    id_list = [{'id': q.id} for q in qset]
    formset = StepFormSet(initial=initial_data)
    if request.method == 'POST':
        formset = StepFormSet(request.POST, initial=initial_data)
        if formset.is_valid():
            models.save_step_formset_with_ids(formset, id_list)
        return redirect('/spec/edit_step_list_use_case/%i/' % use_case.id)
    context = {'formset': formset, 'use_case': use_case, 'step_list':
step_list_with_alt, 'project': project, }
    return render(request, 'spec/edit_step_list_use_case.html', context)

@login_required
def edit_step_list_alt(request, alt_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    alternative = models.get_exception_or_alternative_by_id(alt_id)
```

```
    alternative_list = []
    parent_step = models.get_parent_step_from_alt(alternative.id)
    use_case_id = parent_step.use_case_link_id
    while parent_step.use_case_link_id is None:
        tmp = models.get_parent_alt_from_step(parent_step.id)
        alternative_list.append(tmp)
        parent_step = models.get_parent_step_from_alt(tmp.id)
        use_case_id = parent_step.use_case_link_id
    alternative_list.reverse()
    use_case = models.get_use_case_by_id(use_case_id)
    StepFormSet = formset_factory(form=EditStepForm, extra=0)
    qset = models.get_step_list_from_alt(alternative.id)
    step_list_with_alt = models.step_list_with_alts(qset)
    initial_data = [{'text': q.text, 'number': q.number} for q in qset]
    id_list = [{'id': q.id} for q in qset]
    formset = StepFormSet(initial=initial_data)
    if request.method == 'POST':
        formset = StepFormSet(request.POST, initial=initial_data)
        if formset.is_valid():
            models.save_step_formset_with_ids(formset, id_list)
        return redirect('/spec/edit_step_list_alt/%i/' % alternative.id)
    context = {'formset': formset,
               'alternative': alternative,
               'nav_list': alternative_list,
               'step_list': step_list_with_alt,
               'use_case': use_case,
               'project':
models.get_project_by_id(use_case.project_link_id),
               }
    return render(request, 'spec/edit_step_list_alt.html', context)

@login_required
def delete_step(request, step_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    step = models.get_step_by_id(step_id)
    if request.method == 'POST':
        if step.use_case_link_id is None:
            alternative = models.get_parent_alt_from_step(step.id)
            models.delete_step(step_id)
            return redirect('/spec/edit_step_list_alt/%i/' %
alternative.id)
        else:
            use_case = models.get_use_case_by_id(step.use_case_link_id)
            models.delete_step(step_id)
            return redirect('/spec/edit_step_list_use_case/%i/' %
use_case.id)
    else:
        form = EditStepForm(initial={'number': step.number, 'text':
step.text})
        parent_is_use_case = True
        if step.use_case_link_id is None:
            parent_alt = models.get_parent_alt_from_step(step.id)
            back_id = parent_alt.id
            parent_is_use_case = False
        else:
            back_id = step.use_case_link_id
        context = {'step': step, 'form': form, 'back_id': back_id,
'parent_is_use_case': parent_is_use_case}
        return render(request, 'spec/delete_step.html', context)
```

```
@login_required
def add_alternative_or_exception(request, step_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    step = models.get_step_by_id(step_id)
    if request.method == 'POST':
        form = EditAltForm(request.POST)
        if form.is_valid():
            models.add_exception_or_alternative(form.cleaned_data['code'],

form.cleaned_data['description'], form.cleaned_data['name'], step.id)
            if step.use_case_link_id is None:
                alt = models.get_parent_alt_from_step(step.id)
                return redirect('/spec/edit_step_list_alt/%i/' % alt.id)
            else:
                return redirect('/spec/edit_step_list_use_case/%i/' %
step.use_case_link_id)
    else:
        form = EditAltForm()
        step_have_alt = models.step_have_alternatives(step.id)
        parent_have_alt = models.parent_have_alternatives(step.id)
        parent_is_use_case = True
        if step.use_case_link_id is None:
            parent_alt = models.get_parent_alt_from_step(step.id)
            back_id = parent_alt.id
            parent_is_use_case = False
        else:
            back_id = step.use_case_link_id
        context = {'form': form, 'step': step, 'step_have_alt':
step_have_alt, 'parent_have_alt': parent_have_alt,
                   'back_id': back_id, 'parent_is_use_case':
parent_is_use_case}
        return render(request, 'spec/add_alternative_or_exception.html',
context)

@login_required
def ext_description(request, extension_id):
    extension = models.get_extension_by_id(extension_id)
    use_case = models.get_use_case_by_id(extension.use_case_link_id)
    step = models.get_step_by_id(extension.step_link_id)
    context = {}
    context['extension'] = extension
    context['use_case'] = use_case
    parent_is_use_case = True
    if step.use_case_link_id is None:
        parent_alt = models.get_parent_alt_from_step(step.id)
        back_id = parent_alt.id
        parent_is_use_case = False
    else:
        back_id = step.use_case_link_id
    nav_list = []
    parent_step = step
    use_case_id = parent_step.use_case_link_id
    while parent_step.use_case_link_id is None:
        tmp = models.get_parent_alt_from_step(parent_step.id)
        nav_list.append(tmp)
        parent_step = models.get_parent_step_from_alt(tmp.id)
        use_case_id = parent_step.use_case_link_id
    parent_use_case = models.get_use_case_by_id(use_case_id)
```

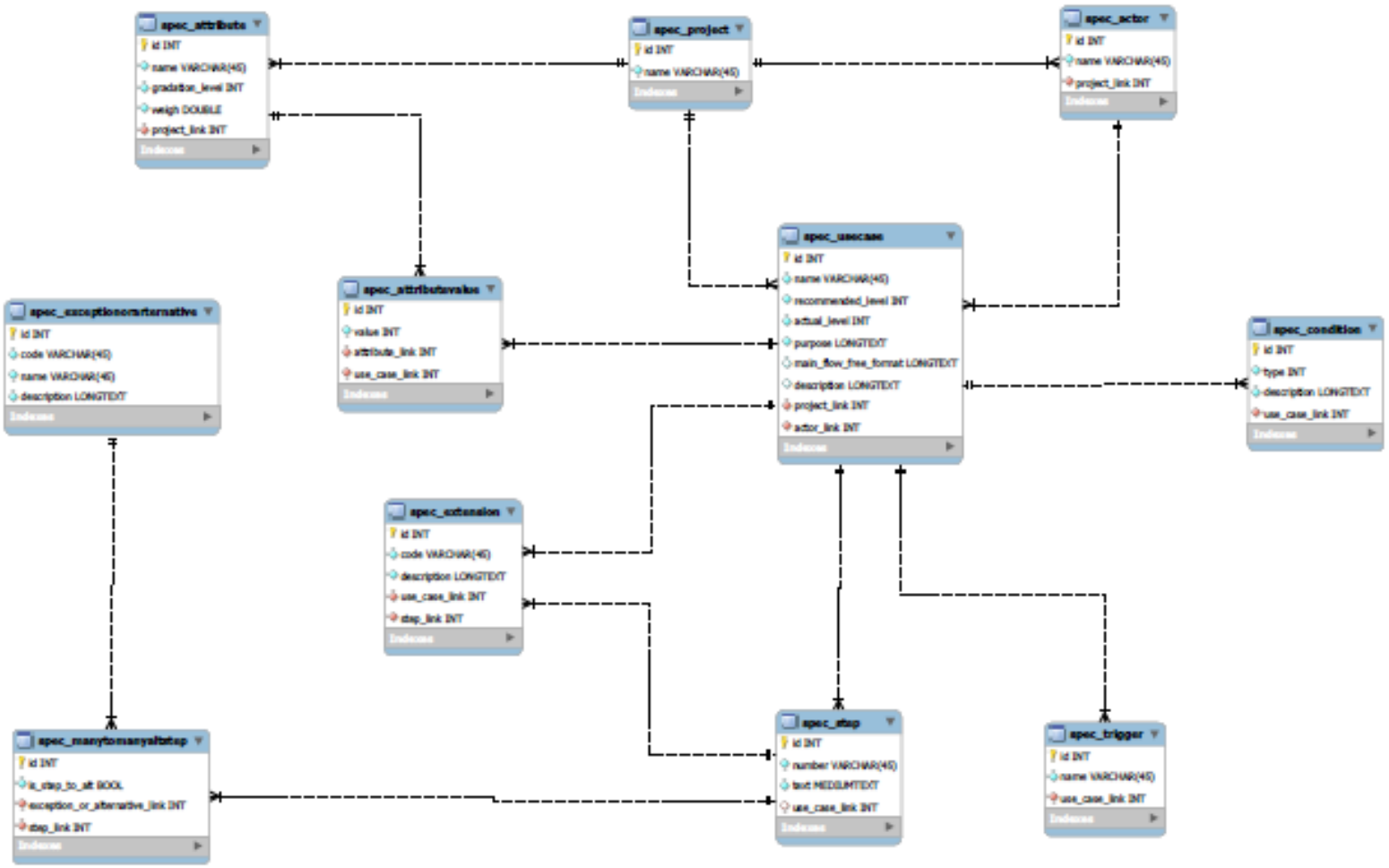
```
    nav_list.reverse()
    project = models.get_project_by_id(use_case.project_link_id)
    context['back_id'] = back_id
    context['parent_is_use_case'] = parent_is_use_case
    context['nav_list'] = nav_list
    context['project'] = project
    context['parent_use_case'] = parent_use_case
    return render(request, 'spec/ext_description.html', context)

@login_required
def add_extension(request, step_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    step = models.get_step_by_id(step_id)
    tmp_step = step
    while tmp_step.use_case_link_id is None:
        parent_alt = models.get_parent_alt_from_step(tmp_step.id)
        tmp_step = models.get_parent_step_from_alt(parent_alt.id)
    use_case = models.get_use_case_by_id(tmp_step.use_case_link_id)
    use_case_list = models.get_use_case_list(use_case.project_link_id)
    if request.method == 'POST':
        form = ChooseUseCase(request.POST)
        form.fields['use_case'].choices = [(q.id, q.name) for q in
use_case_list]
        if form.is_valid():
            models.add_extension(
                step_id, form.cleaned_data['use_case'],
form.cleaned_data['code'], form.cleaned_data['description'])
            if step.use_case_link_id is None:
                parent_alt = models.get_parent_alt_from_step(step.id)
                return redirect('/spec/edit_step_list_alt/%i/' %
parent_alt.id)
            else:
                use_case = models.get_use_case_by_id(step.use_case_link_id)
                return redirect('/spec/edit_step_list_use_case/%i/' %
use_case.id)
    else:
        form = ChooseUseCase()
        parent_is_use_case = True
        if step.use_case_link_id is None:
            parent_alt = models.get_parent_alt_from_step(step.id)
            back_id = parent_alt.id
            parent_is_use_case = False
        else:
            back_id = step.use_case_link_id
        form.fields['use_case'].choices = [(q.id, q.name) for q in
use_case_list]
        context = {'form': form, 'step': step, 'back_id': back_id,
'parent_is_use_case': parent_is_use_case}
        return render(request, 'spec/add_extension.html', context)

@login_required
def edit_extension(request, extension_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    extension = models.get_extension_by_id(extension_id)
    step = models.get_step_by_id(extension.step_link_id)
    use_case = models.get_use_case_by_id(extension.use_case_link_id)
    if request.method == 'POST':
```

```
        form = EditAltForm(request.POST)
        if form.is_valid():
            models.edit_extension(
                extension.id, form.cleaned_data['code'],
form.cleaned_data['description'])
            if step.use_case_link_id is None:
                parent_alt = models.get_parent_alt_from_step(step.id)
                return redirect('/spec/edit_step_list_alt/%i/' %
parent_alt.id)
            else:
                return redirect('/spec/edit_step_list_use_case/%i/' %
use_case.id)
    else:
        nav_list = []
        parent_step = step
        use_case_id = parent_step.use_case_link_id
        while parent_step.use_case_link_id is None:
            tmp = models.get_parent_alt_from_step(parent_step.id)
            nav_list.append(tmp)
            parent_step = models.get_parent_step_from_alt(tmp.id)
            use_case_id = parent_step.use_case_link_id
        parent_use_case = models.get_use_case_by_id(use_case_id)
        nav_list.reverse()
        form = EditAltForm(initial={'code': extension.code, 'description':
extension.description})
        project = models.get_project_by_id(use_case.project_link_id)
        context = {'form': form, 'extension': extension, 'nav_list':
nav_list,
                   'use_case': use_case, 'project': project,
'parent_use_case': parent_use_case}
        return render(request, 'spec/edit_extension.html', context)

@login_required
def delete_extension(request, extension_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    extension = models.get_extension_by_id(extension_id)
    step = models.get_step_by_id(extension.step_link_id)
    if request.method == 'POST':
        models.delete_extension(extension.id)
        if step.use_case_link_id is None:
            alternative = models.get_parent_alt_from_step(step.id)
            return redirect('/spec/edit_step_list_alt/%i/' %
alternative.id)
        else:
            use_case = models.get_use_case_by_id(step.use_case_link_id)
            return redirect('/spec/edit_step_list_use_case/%i/' %
use_case.id)
    else:
        context = {'extension': extension}
        return render(request, 'spec/delete_extension.html', context)

@login_required
def delete_extension_by_step(request, step_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    step = models.get_step_by_id(step_id)
    use_case_list = models.get_extension_list_by_step(step.id)
    if request.method == 'POST':
```

```
        form = ChooseSomething(request.POST)
        form.fields['something'].choices = [(q.id, q.code) for q in
use_case_list]
        if form.is_valid():

models.delete_extension_with_find(form.cleaned_data['something'], step.id)
            if step.use_case_link_id is None:
                parent_alt = models.get_parent_alt_from_step(step.id)
                return redirect('/spec/edit_step_list_alt/%i/' %
parent_alt.id)
            else:
                use_case = models.get_use_case_by_id(step.use_case_link_id)
                return redirect('/spec/edit_step_list_use_case/%i/' %
use_case.id)
    else:
        form = ChooseSomething()
        form.fields['something'].choices = [(q.id, q.code) for q in
use_case_list]
        context = {'form': form, 'step': step}
        return render(request, 'spec/link_alternative.html', context)

@login_required
def add_step_u(request, use_case_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    use_case = models.get_use_case_by_id(use_case_id)
    number = models.get_step_number_from_use_case(use_case_id)
    models.add_step(number + 1, ' ', use_case_id)
    return redirect('/spec/edit_step_list_use_case/%i/' % use_case.id)

@login_required
def add_step_a(request, alt_id):
    user = auth.get_user(request)
    if not user.is_staff:
        return redirect('/spec/permission_error/')
    alternative = models.get_exception_or_alternative_by_id(alt_id)
    number = models.get_step_number_from_alt(alternative.id)
    models.add_step_from_alt(number + 1, ' ', alternative.id)
    return redirect('/spec/edit_step_list_alt/%i/' % alternative.id)

def main(request):
    user = auth.get_user(request)
    if user.is_authenticated:
        return redirect('/spec/project_list/')
    else:
        return redirect('/accounts/login/')
    context['user'] = user
    return render(request, 'spec/main.html', context)

def permission_error(request):
    return render(request, 'spec/permission_error.html')

def logout_c(request):
    auth.logout(request)
    return redirect('/accounts/login/')
```

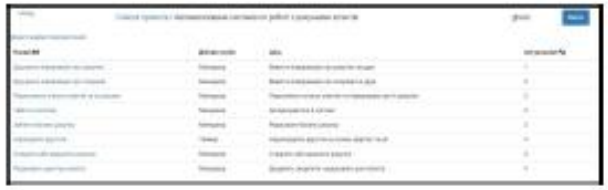
Додаток 2

Копії графічних матеріалів



Шабіневич Роман, КП-81мп

Перший рівень



Другий рівень

Третій рівень

Четвертий рівень

Шабіневич Роман, КП-81мп

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

СПОСІБ АВТОМАТИЗАЦІЇ ПРОЦЕСУ СТВОРЕННЯ СПЕЦИФІКАЦІЙ ВАРІАНТІВ ВИКОРИСТАННЯ НА ОСНОВІ ЇХ АТРИБУТІВ

Виконав: Шабіневич Роман Олександрович

Науковий керівник: к.т.н. Рибачок Наталія Антонівна

Київ – 2019

АКТУАЛЬНІСТЬ

- Специфікації ВВ використовуються при розробленні ПЗ, особливо коли команда територіально розділена
- Відсутність на ринку програмних рішень з гнучким налаштуванням для специфікації варіантів використання
- Процес специфікації ВВ з використанням методу Коберна (та подібних) можна автоматизувати та підвищити гнучкість алгоритму вибору рівня деталізації з мінімальним збільшенням навантаження на працівника

Об’єкт дослідження: процес специфікації варіантів використання

Предмет дослідження: спосіб автоматизації процедури специфікації варіантів використання

Наукове завдання: автоматизувати процес специфікації варіантів використання на основі їх атрибутів.

Мета дослідження: розроблення способу автоматизації процесу створення специфікацій варіантів використання на основі їх атрибутів.

Задачі

Відповідно до вказаної мети необхідно розв’язати такі задачі:

- проаналізувати існуючі підходи до специфікації ВВ;
- сформулювати узагальнений спосіб вибору рівня деталізації специфікацій ВВ на основі їх атрибутів;
- проаналізувати існуючі шаблони специфікацій ВВ та створити набір шаблонів для нарощування рівня деталізації та можливості модульного подання;
- реалізувати спосіб автоматизації процесу створення специфікацій варіантів використання.

Для чого потрібні варіанти використання?

Контракт між розробниками і замовниками

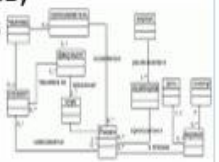
Основа для проектування системи:

- обмеження
- формати даних
- функціональні вимоги
- нефункціональні вимоги

Можна отримувати документацію для користувачів

Варіант використання

Можна створювати діаграми послідовностей, діаграми зв'язків, діаграми класів



Можна використовувати в якості вхідних даних для тестових сценаріїв (test cases)

Корисні в плануванні формального змісту ітерацій; дозволяють розробникам краще зрозуміти призначення системи

Коли потрібна специфікація варіантів використання?

Потрібна. Якщо в системі:

- переважають функціональні вимоги
- багато типів користувачів з різними цілями
- багато інтерфейсів
- автоматизуються бізнес-процеси

Не потрібна. Якщо в системі:

- переважають нефункціональні вимоги
- мало користувачів і інтерфейсів
- інтеграційні проекти

Процес специфікації варіантів використання (за Коберном)

1. Виявити всі можливі ВВ, які прийняті для реалізації в ПЗ
2. Оформити виявлені варіанти використання у Реєстр ВВ
3. Визначити рівень специфікації кожного ВВ експертним шляхом
4. Написати специфікації

Модифікація способу визначення рівня деталізації

Узагальнений спосіб вибору рівня деталізації специфікацій ВВ враховує підтримку основних методик пріоритизації вимог та базується на основі формули середньої арифметичної зваженої. Її математичне представлення наведено нижче:

$$\mathrm{f}(\mathrm{A1},\ \ldots, An) = \left\lfloor \frac{\sum(Ci * Ai * \frac{4}{Cgi})}{\sum Ci} \right\rfloor,$$

де Cgi – ступінь градації i-го атрибуту

Ai – значення i-го атрибуту (1- Cgi)

Ci – значення вагового коефіцієнту для i-го атрибуту

Приклад роботи способу

Для автоматизованої системи по роботі з рахунками клієнтів визначено набір з 8 основних варіантів використання

Спрощений реєстр ВВ

№	Назва ВВ
UC1	Друкувати інформацію про операції
UC2	Друкувати інформацію про рахунок
UC3	Змінити баланс рахунку
UC4	Нарахувати відсотки
UC5	Переглянути список клієнтів та їх рахунки
UC6	Редагувати дані по клієнта (CRUDL)
UC7	Створити або видалити рахунок
UC8	Увійти в систему

Приклад роботи способу

i	Атрибут	Зміст атрибуту	Ci	Cgi
1	Priority (MoSCoW)	рівень зацікавленості дійових осіб у реалізації ВВ	0,7	4
2	Difficulty (RUP)	рівень складності реалізації	1	3

№	Priority	Difficulty	Рекомендований рівень деталізації
UC6	Must (4)	High (3)	4
UC3	Could (2)	High (3)	3
UC7	Must (4)	Medium (2)	3
UC4	Won't (1)	High (3)	3
UC5	Must (4)	Low (1)	2
UC1	Won't (1)	Medium (2)	2
UC8	Should (3)	Low (1)	2
UC2	Won't (1)	Low (1)	1

Перший рівень деталізації

- Короткий опис в реєстрі варіантів використання

№	Назва ВВ	Дійова особа	Ціль	P	D

Другий рівень деталізації

- Загальний короткий опис (назва ВВ та дійові особи)
- Короткий опис основного потоку в довільній текстовій формі

Третій рівень деталізації

- Загальний короткий опис
- Покроковий опис основного потоку
- Передумови, післяумови, спеціальні умови
- Перелік альтернатив та виключних ситуацій

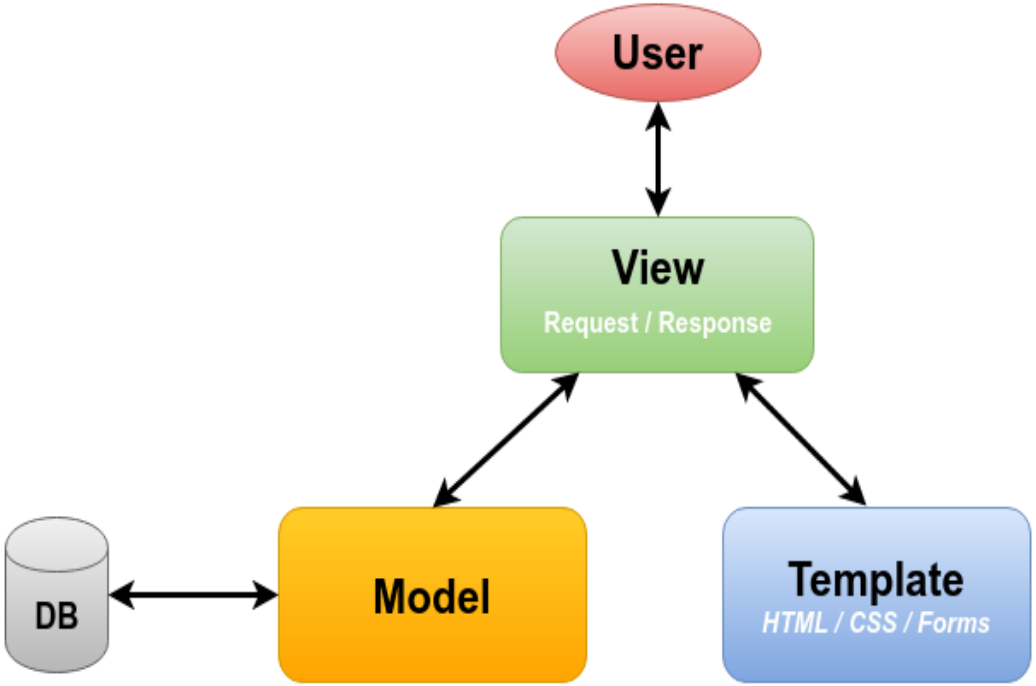
Четвертий рівень деталізації

- Загальний короткий опис
- Покроковий опис основного потоку
- Передумови, післяумови, спеціальні умови
- Покроковий опис альтернатив та виключних ситуацій
- Точки розширення

Переваги модульного подання специфікацій

- Просте нарощування рівня деталізації
- Зниження інформаційного навантаження при роботі користувача
- Локалізація змін в специфікаціях

Архітектура розробленого застосунку



Використані технології

Науково-технічна новизна

Запропонований спосіб автоматизації процесу специфікації варіантів використання відрізняється від аналогів тим, що дозволяє комбінувати атрибути з різних методик та працювати з специфікаціями в модульному вигляді

Бізнес модель

<table>
<tr><th>Проблема</th><th>Рішення</th><th>Унікальна ціннісна пропозиція</th><th>Приховані переваги</th><th>Споживачі</th></tr>
<tr><td rowspan="2">Надмірна кількість паперової та ручних розрахунків при специфікації варіантів використання</td><td>Розроблення програмного забезпечення для автоматизації процесу специфікації ВВ</td><td rowspan="2">• Автоматизація процесу написання специфікацій.
• Зручний інтерфейс.
• Легкість використання</td><td>• Веб-орієнтованість
• Один примірник на одну компанію підвищує захищеність даних</td><td rowspan="2">Компанії, які займаються розробленням програмного забезпечення</td></tr>
<tr><td>Ключові показники
Кількість проданих ліцензій</td><td>Канали
Відділи інтеграції компаній, які виконують розроблення ПЗ</td></tr>
<tr><th colspan="3">Структура витрат</th><th colspan="2">Потоки доходів</th></tr>
<tr><td colspan="3">• Проведення науково-дослідних та дослідно-конструкторських робіт
• Оплата праці персоналу
• Податкові витрати</td><td colspan="2">• Доходи від продажу ліцензій програмного забезпечення
• Доходи від підтримки програмного забезпечення</td></tr>
</table>

Фінансовий план стартапу

(тис. дол. США)	Загальні витрати	Оплата праці	Підсумок витрат	Заплановані прибутки	Результат (без оподаткування)
Сумарно за рік:	**51**	**112**	**163**	**240**	**69**

Апробація

- XII наукова конференція магістрантів та аспірантів «Прикладна математика та комп’ютинг» (ПМК-2019)

Висновки

Практичне застосування результатів даного дослідження дозволить:

1. зменшити часові витрати на створення специфікацій ВВ (за рахунок автоматичного вибору рівня деталізації та підтримки відповідного шаблону);
2. полегшити супровід специфікацій після їх створення (як при змінах у описах, так і при змінах у рівнях деталізації);
3. уніфікувати процес документування специфікацій ВВ за рахунок використання шаблонів.

Дякую за увагу